AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI

Koncepcja Kaskadowych gramatyk kształtu jako modelu przetwarzania kształtów graficznych w systemach bazujących na zastosowaniu metod półformalnych.

mgr inż. Michał Turek

Rozprawa doktorska przygotowana pod kierownictwem

dr hab. inż. Tadeusza Szuby

Kraków 2010

Chciałbym bardzo serdecznie podziękować
Panu dr hab. inż. Tadeuszowi Szubie
za wszystkie wskazówki i rady
stanowiące pomoc podczas tworzenia tej pracy.

Zawartość pracy

1 Wprowadzenie

Ciągły wzrost strukturalnej złożoności komponentów graficznych wykorzystywanych przy renderowaniu lub projektowaniu jakichkolwiek graficznych obiektów trójwymiarowych jest wyraźnie zauważalny i oczywisty. Niezależnie od dziedziny w jakiej zawiera się tematyka dotycząca projektowanych komponentów graficznych oraz docelowej technologii ich późniejszego prezentowania (trójwymiarowe interaktywne prezentacje, renderowanie statycznych scen dwu- lub trójwymiarowych, zastosowania CAD itp.) przed osiągnięciem końcowego produktu wymagane są żmudne i pracochłonne czynności przygotowawcze. Mają na celu wytworzenie dużej ilości materiałów będących komponentami końcowego produktu. Materiały te to głównie graficzne reprezentacje obiektów fizycznych oraz transformacje wykorzystywane do prowadzenia renderowanej w czasie rzeczywistym trójwymiarowej i dynamicznej projekcji tych obiektów. Skłania to do refleksji nad możliwościami wprowadzenia procesów automatyzujących te wydłużające się czynności przygotowawcze. Ogromne zróżnicowanie zagadnień, pośród których znajdują zastosowanie zautomatyzowane graficzne metody projektowe lub symulacycji zachęca do poszukiwania jedynie rozwiązań uniwersalnych – abstrahujących od przeznaczenia powstałego produktu czy ograniczeń narzucanych przez świat, w których funkcjonują obrazowane graficznie obiekty.

Celem niniejszej pracy jest wykazanie, iż przy zastosowaniu odpowiedniej metodologii możliwe jest tworzenie systemów zarządzających procesami generowania kształtów graficznych w sposób całkowicie niezależny od dziedziny tych kształtów i jednocześnie w pełni autonomiczny. Głównym nurtem tematycznym pracy są rozważania na temat możliwości pół-formalnego określania procesów transformacji obiektów w przestrzeni trójwymiarowej (3D). Przy obecnym stanie wiedzy na temat dostępnych już transformacji obiektów trójwymiarowych powinna istnieć możliwość odwołania się do metod nietypowych oraz mających podłoże interdyscyplinarne. Otworzy to nowe drogi do rozwoju rozwiązań wspomagających modelowanie trójwymiarowe. Potencjalnie opracowane rozwiązania mogą zostać zastosowanie wszędzie tam, gdzie zaistnieje potrzeba prowadzenia prac projektowych czy tworzenia prostych symulacji w oparciu o określony strukturalnie model opisujący przestrzeń 3D oraz o wiedzę eksperta w dziedzinie, z której model się wywodzi.

1.1 Zawartość pracy

Praca podzielona została na osiem głównych rozdziałów (interpretując łącznie z rozdziałem bieżącym i podsumowaniem).

W rozdziale następnym (**drugim**) zostało zamieszczone teoretyczne wprowadzenie do zagadnień związanych z przedmiotami pracy. Rozdział prezentuje definicję gramatyk kształtu, ich genezę oraz wstępne rozważa teoretycznie na temat możliwości zastosowania metod półformalnych, a w tym gramatyk kształtu na gruncie praktycznym i naukowym. Rozdział przybliża podstawy gramatyk kształtu, omawia także teorię samych transformacji kształtów w przestrzeni trójwymiarowej. Zawiera również przegląd innych metod formalnych, mogących mieć znaczenie dla rozważanej później materii.

Rozdział **trzeci** zawiera przegląd znanych obecnie osiągnięć mających związek z automatyzacją procesów modelowania przestrzeni 3D przy wykorzystaniu niektórych systemów generacyjnych i innych rozwiązań pokrewnych. Większość leżących w kręgu zainteresowania autora metod dotyczy przetwarzania wiedzy o transformacjach 3D opartego na regułach. Takie rozwiązania zostały tu zanalizowane za szczególną dokładnością. Powodem większego zainteresowania jest powinowactwo reguł wnioskowania (stanowiących środek do prowadzenia analizy na materiale uporządkowanym dziedzinowo w klasycznych systemach ekspertowych) do produkcji będących elementami zbioru gramatyki kształtu (umożliwiających przetwarzanie materiału w graficznych systemach generacyjnych). Szczególną uwagę poświęcono pracom tych zespołów badawczych, które dokonały prób rozwinięcia formalnej definicji gramatyki kształtu o nowe konstrukcje w różnych zastosowaniach.

Rozdział **czwarty** zawiera podsumowanie rozważań dotyczących stanu wiedzy w zakresie wykorzystania gramatyk kształtu oraz rozważa potencjalne możliwości jej wzbogacenia w kierunkach zgodnych z obraną tematyką pracy. Formułuje następnie cele pracy i tezę, rozbudowaną także o tezy poboczne.

Rozdział **piąty** to opis teoretycznych rozważań nad elementami metodologii prowadzącej do wsparcia tez pracy. Bazując na zebranej wcześniej wiedzy przedstawia on propozycję wprowadzenia metodologii tzw. Kaskadowych gramatyk kształtu. Opisuje on kroki, jakie zostały podjęte w celu opracowania wspomnianej metodologii. Prezentuje proponowany wielowarstwowy model przetwarzania z udziałem takiej gramatyki, charakteryzując także liczne nowo zdefiniowane pojęcia i rozwiązania. W rozdziale oszacowana zostanie przestrzeń

funkcjonowania Kaskadowych gramatyk kształtu, określając naturę przetwarzanego materiału, jego potencjalne źródła oraz metody wymiany wiedzy o nim.

Rozdział **szósty** zawiera dalsze opisy i analizy przyjętych rozwiązań. Prowadzi szeroki wywód na temat proponowanych w pracy metodyk budowania systemów przetwarzających materiały graficzne w oparciu o zaproponowany model Kaskadowych gramatyk kształtu. Opisuje proponowaną funkcjonalność komponentów systemu, zlokalizowanych w poszczególnych warstwach. Charakteryzuje także interfejsy systemu, w tym umożliwiajace operowanie na globalnej przestrzni komponentów graficznych z udziałem wymiany ontologicznej. W rozdziale opisane zostanie także oprogramowanie, jakie tworzone było na poszczególnych etapach prac badawczych. Przedstawione zostaną zarówno moduły wspomagające transformacje z wykorzystaniem metodologii Kaskadowych gramatyk kształtu jak i oprogramowanie służące do prowadzenia testów oraz realizacji procesu wysokowydajnej wizualizujacji wyników poszczególnych eksperymentów.

Rozdział **siódmy** stanowi omówienie eksperymentów prowadzonych z użyciem systemów opartych na przetwarzaniu materiału graficznego Kaskadowymi gramatykami kształtu. Na bazie przykładów konkretnych gramatyk prezentuje on spektrum możliwości wykorzystania takich systemów w różnorodnych zagadnieniach praktycznych.

W rozdziale tym opisana zostanie analiza cech poza funkcjonalnych rozwiązań, zorientowana wokół testów wydajnościowych algorytmów prowadzących procesy transformacji. Opisane zostaną udane próby integracji wysokowydajnych systemów generacyjnych opartych o wspomnianą metodologię ze wspomaganymi sprzętowo trójwymiarowymi graficznymi platformami renderującymi.

Rozdział **ostatni** przedstawia podsumowanie rozważań i uzyskanych wyników wspierających prawdziwość postawionych w pracy tez.

2 Podstawowe zagadnienia

Treść rozdziału drugiego stanowi wprowadzenie do zagadnień określających podstawy teoretyczne rozważań nad metodykami tworzenia graficznych systemów generacyjnch. Uporządkowane tu zostaną definicje pojęć związanych z tymi zagadnieniami. Tak przeprowadzone ugruntowanie teoretyczne jest konieczne do rozpoczęcia analiz osiągnięć dokonanych już w dziedzinie rozbudowy graficznych systemów generacyjnych oraz do prowadzenia analiz możliwości ich ulepszania.

2.1 Ogólna charakterystyka dziedziny pracy

Na przestrzeni ostatnich kilkunastu lat dokonywano wielu prób zautomatyzowania czynności projektowych w trójwymiarowej grafice komputerowej. Dążono jednak głównie do wytworzenia gotowych zastosowań w oprogramowaniu dla wąskich i wyspecjalizowanych dziedzin. Przyjmowano najróżniejsze metodyki – od bazujących na zapisie formalnym po całkowicie heurystyczne. Problemem stojącym na przeszkodzie do użycia metod formalnych w zastosowaniach inżynieryjnych rozszerzalnych na wiele różnorodnych dziedzin modelowania, projektowania czy renderowania grafiki komputerowej jest konieczność formalnego opisu precyzyjnych wymagań stawianych dla konkretnego opracowania inżynieryjnego. W przypadku grafiki najczęściej nie jest to możliwe. Wymagania stawiane wynikom nie mogą być upraszczane (poprzez eliminację detali, fragmentację) jedynie do opisu reprezentacji symbolicznej spodziewanego wyniku. Utworzenie pomostu pomiędzy terminologią, w której operuje projektant obiektów w wirtualnej przestrzeni graficznej, a materiałem zapisanym formalnie jest niewątpliwie dużym wyzwaniem. Można stwierdzić, iż brak jest metodologii pozwalającej jednocześnie na:

- opisanie różnorodnych relacji wiążących znaczenie obiektów graficznych będących elementami wirtualnej przestrzeni graficznej odzwierciedlającej daną dziedzinę świata rzeczywistego,
- wykorzystanie tych relacji w kontrolowanym procesie półformalego przetwarzania obiektów graficznych w trójwymiarowej wirtualnej przestrzeni,

- dodatkowe uwzględnienie w procesie przetwarzania cech geometrycznych obiektów istniejących w trójwymiarowej wirtualnej przestrzeni,
- zaangażowanie metod formalnych do rozbudowy reprezentacji graficznych obiektów tej przestrzeni,
- łatwe i intuicyjne pobranie wiedzy na temat komponentów graficznych od eksperta w dziedzinie.

Problem powodowany przez tą ułomność stanowi barierę na drodze rozwoju systemów wspomagających projektowanie trójwymiarowych obiektów graficznych. Linia przewodnia, wzdłuż której przebiega szkic proponowanego w pracy rozwiązania bazuje na koncepcji utworzenia modelu, który zaangażuje uniwersalne (ogólnie ujęte) i formalnie zdefiniowane reguły przetwarzania. Jednocześnie będzie agregował różnorodne i już wyspecjalizowane dziedzinowo transformacje do obrabiania konkretnego materiału graficznego. Model ten będzie posiadał architekturę warstwową. Transformacje, zależne od reprezentacji graficznej materiału, jego przeznaczenia oraz szeregu dalszych uzależnień dyktowanych każdorazowo otrzymywanym zadaniem, będą określały konkretne elementarne czynności edycyjne. Takie, które są możliwe do przeprowadzania na materiale graficznym. Wykorzystanie funkcjonalności dostarczonej przez transformacje będzie leżało w gestii systemu decyzyjnego, opartego na formalnie zdefiniowanej i częściowo abstrakcyjnej wiedzy.

W toku doświadczeń zostanie wykazane, iż rozdzielenie abstrakcji obiektu należącego do określonej dziedziny zagadnień od konkretnej reprezentacji graficznej tego obiektu pozwala na precyzyjne zdefiniowanie przestrzeni działania regułowego systemu sterującego użytkowaniem transformacji graficznych w konkretnych sytuacjach. Przestrzeń ta będzie wówczas logicznie niezależna od przekształcanych przez system obiektów, które teoretycznie mogą nawet nie posiadać wspomnianej reprezentacji graficznej (nie być obiektami graficznymi).

Podstawową motywacją dla rozważań nad wspomnianym problemem jest ciągle rosnące zapotrzebowanie na dużą ilość materiałów 3D tworzonych głównie na potrzeby animacji trójwymiarowych. Stwarza to podstawę do zastanowienia nad projektowaniem metodyki zautomatyzowanego ich wytwarzania. Tradycyjne metody (czyli głównie edycja manualna) wymagają w swoich zastosowaniach zaawansowanych i interaktywnych edytorów 3D absorbując jednocześnie w dużych ilościach czas oraz uwagę projektanta. Większość modeli 3D będących komponentami określonego produktu komercyjnego posiada liczne cechy wspólne i

jest wytwarzana w sposób schematyczny - z wyraźnymi analogiami w ich architekturze. Tym samym czynności manualne przy ich tworzeniu często się powtarzają. Z drugiej strony – już istniejące rozwiązania zautomatyzowane zakładają z reguły istnienie bardzo skąpo opisanego świata kształtów 3D, wobec których dana metoda może być zastosowana. Redukuje to jej przydatność właściwie jedynie do teoretycznych zastosowań akademickich. To tworzy podstawę do rozważania nad poszukiwaniami alternatywnych dróg pozyskiwania takich materiałów.

Celem pracy jest poszukiwanie nowych metod pozyskiwania skomplikowanych i spełniających złożone wymagania obiektów graficznych – z postawieniem nacisku na rozszerzalność tych metod, ich uniwersalność i łatwość uwidocznienia ich zalet w postaci konkretnych rozwiązań inżynieryjnych.

Główna ścieżka projektowa będzie dążyć do opracowania metodyki, na bazie której możliwe będzie tworzenie automatów generujących obiekty graficzne w przestrzeni n-wymiarowej. Przeprowadzona zostanie rozległa ewaluacja znanych obecnie mniej lub bardziej wartościowych metod automatycznego generowania kształtów graficznych - z wytypowaniem opracowań przydatnych od strony metodycznej, czyli dostarczających wskazówek jak wartościowy system przetwarzający grafikę 3D należy projektować. Szczególny nacisk położony zostanie ma metodyki postulujące wykorzystanie wiedzy opartej na regułach.

Ze szczególną uwagą prześledzone zostaną założenia znanej od prawie 40 lat teorii *gramatyk kształtu*, której wybór jako podstawy teoretycznej do licznych pokrewnych opracowań w omawianej problematyce był wielokrotnie dokonywany. Przesłanki do takiego wyboru są liczne. Spośród nich warto szczególnie przedłożyć następujące:

- gramatyki kształtu posiadają ścisłą definicję formalną,
- wszelkie elementy definiowane jako komponenty gramatyki kształtu to zbiory,
- gramatyki kształtu doczekały się licznych ulepszających je kontynuacji,
- aparat matematyczny wykorzystywany w procesie generowania kształtów graficznych z użyciem gramatyk kształtu jest łatwy do rozwinięcia.

Pośrednim celem opracowania będzie rozwinięcie teorii gramatyk kształtu do zupełnie nowej postaci - uzupełniając jej model o liczne mechanizmy dające możliwość tworzenia konkretnych transformacji zdatnych do praktycznego i efektywnego wykorzystania.

Opracowana przez autora nowa koncepcja gramatyk kształtu zostanie nazwana "*Kaskadowymi gramatykami kształtu*", określnymi w pracy roboczo akronimem K-GK.

2.2 Przekształcenia w przestrzeni trójwymiarowej – uporządkowanie pojęć

W ostatnich latach w rozmaitych publikacjach poruszających zagadnienia związane z przetwarzaniem obiektów graficznych dość swobodnie posługiwano się rozmaitą terminologią określającą wszelkie obiekty będące przedmiotem prezentacji czy transformacji graficznej. Na potrzeby niniejszej pracy konieczne będzie uporządkowanie wspomnianych pojęć i doprecyzowanie licznych definicji. Dla zwiększenia przejrzystości każde odwołanie do zdefiniowanego tutaj terminu i występujące w późniejszych rozdziałach pracy będzie oznaczone tekstem *pochylonym*.

W pierwszej kolejności należy zdefiniować pojęcie *kształtu*. W publikacjach występuje ono licznie, reprezentując słownikowe znaczenie terminu „wygląd", „opis", „forma" etc. Pojęcie to w wielu przypadkach w ogóle nie jest definiowane lecz klasyfikowane jako pierwotne. Należy je jednak dookreślić.

Definicja 2.1:

> *Kształtem* w rozumieniu ogólnym będziemy nazywali zbiór dowolnych identyfikowalnych obiektów, posiadających atrybuty, usytuowanych w dowolnej przestrzeni [Sti06]. ■

Przetwarzanie zestawu kształtów przy użyciu gramatyki (operującej na symbolach) będzie wymagało przypisywania określonych symboli do kształtów. Tu pomocny będzie termin *semantyki* – rozumiany w pracy klasycznie, czyli jako przypisanie elementom języka obiektów z jakiejś dziedziny [Maj80].

Kontynuacją pojęcia kształtu będzie „kształt geometryczny" lub często stosowany jego zamiennik - „kształt graficzny". Ten termin zawęża definicję kształtu do puli kształtów mających postać graficzną. Spotykamy go w przypadku rozważań dotyczących samej grafiki komputerowej. Definicja kształtu geometrycznego na potrzeby rozważań w pracy zostanie ujęta następująco:

Definicja 2.2:

> *Kształt graficzny (geometryczny)* to zbiór dowolnych identyfikowalnych obiektów, posiadających atrybuty i usytuowanych w *n*-wymiarowej przestrzeni Euklidesa (euklidesowej). W wymiarze praktycznym n będzie wynosić 2 lub 3. ■

Przestrzeń Euklidesa o wymiarze n równym 2 będziemy dodatkowo nazywali *przestrzenią dwuwymiarową* lub *przestrzenią 2D*. Kształt graficzny zawarty w tej przestrzeni – *kształtem 2D* lub *kształtem dwuwymiarowym*.

Przestrzeń Euklidesa o wymiarze n równym 3 będziemy dodatkowo nazywali *przestrzenią trójwymiarową* lub *przestrzenią 3D*. *Kształt graficzny* zawarty w tej przestrzeni – *kształtem 3D* lub *kształtem trójwymiarowym*.

Gdy mowa będzie o technikach prezentowania obiektów graficznych umieszczonych w trójwymiarowej przestrzeni Euklidesa, przestrzeń tą (przeznaczoną dla wspomnianych obiektów) będziemy określać mianem *przestrzeni projekcji* lub *wirtualnej przestrzeni projekcji.*

Dalszy (pomocniczy) termin, który często jest widywany to *cecha kształtu graficznego*. Identyfikuje on wszelkie informacje wyróżniające kształty i jest tożsamy z możliwymi do zinterpretowania wartościami atrybutu obiektu będącego kształtem. *Cechy kształtów graficznych* są klasyfikowane i w naturalnym trybie stanowią podstawę klasyfikacji *kształtów* w opracowaniach proponujących zaawansowane rozwiązania przekształcające obiekty w przestrzeni. Klasyfikacja *cech kształtów* często opiera się na wprowadzaniu dedykowanych języków opisu kształtów *(SFDL - Shape Feature Description Language)*

Terminem, którym także często trzeba się posługiwać jest tzw. *transformata*. Formalnie *transformata* to wynik przekształcenia *operandu* pod wpływam działania *operatora*. Pozostając w zgodzie z powyższą definicją należy przyjąć, iż w przypadku przekształceń obiektów w grafice komputerowej *operandem* i *transformatą* będą *kształty graficzne*, zaś *operatorami* będą różnorodne algorytmy definiujące przekształcenia w grafice 3D.

Transformacja graficzna będzie zatem procesem, który dostarczy *transformat operandów* (*kształtów graficznych*) w wyniku zastosowania *operatorów*.

2.3 Metody formalne leżące u podstaw gramatyk kształtu

Analiza istniejących już rozwiazań skoncentrowanych wokół technik tworzenia graficznych systemów generacyjnych opartych na regułach wymaga sięgnięcia także do ich podstaw teoretycznych. Podstawy te dotyczą głównie sposobów opisu języka formalnego oraz funkcji elementu probabilistycznego pełnionej w systemach przetwarzających reguły. Rozmiar bieżącego podrozdziału zredukowano do niezbędnego minimum opisując tylko pojęcia, z których czerpią konkretne rozwiązania rozpatrywane w rozdziale trzecim.

2.3.1 Krótkie wprowadzenie do gramatyk formalnych

Systemy języków formalnych umożliwiają jednoznaczny opis symboliczny tworzonych nimi konstrukcji. Niniejszy podrozdział przedstawi krótko najbardziej istotne rodzaje gramatyk leżące także często u podstaw różnorodnych zastosowań w grafice komputerowej. Przybliżając zalety gramatyk formalnych w takich zastosowaniach (wśród których należy przede wszystkim wymienić możliwość prowadzenia jednoznacznego zapisu treści) przygotuje grunt pod wprowadzenie do samych gramatyk kształtu, znajdujących się w centrum tematyki pracy.

Aby opisać jakikolwiek obiekt czy mechanizm związany z gramatykami formalnymi konieczne jest wprowadzenie szeregu pojęć pomocniczych, które w gramatykach funkcjonują. Pojęcia te posłużą także do zbudowania definicji gramatyk formalnych w różnych wariantach. Opisy takich pojęć przedstawia pozostałą część tego podrozdziału:

Symbol jest pojęciem pierwotnym, więc nie definiowalnym. Interpretując rzecz opisowo - symbol, posiadając reprezentację graficzną, jest znakiem, literą, cyfrą lub innym rozpoznawalnym elementarnym kodem.

Alfabet to niepusty i jednocześnie skończony zbiór symboli. Oznaczając (zwyczajowo) alfabet symbolem T można napisać, że $T \neq \varnothing$ oraz $\#T < \infty$, gdzie #T to moc zbioru T.

Słowo alfabetu (lub łańcuch z alfabetu) to skończony ciąg występujących po sobie symboli z alfabetu. Definicję słowa można także ująć następująco:

„Łańcuch pusty jest łańcuchem nad alfabetem T oraz jeżeli x_1 jest łańcuchem nad T i $x_2 \in T$ to x_1x_2 także jest łańcuchem nad T. Dodatkowo – żaden inny łańcuch nie jest łańcuchem nad T.”

Zbiór słów alfabetu to zbiór wszystkich możliwych do ułożenia słów danego alfabetu, znakowany zwyczajowo poprzez T*.

Język nad alfabetem to dowolny podzbiór zbioru słów alfabetu. Gdy przykładowo alfabet zostanie oznaczony literą T, zbiór słów alfabetu literą T* to język L nad alfabetem T będzie spełniał: $L \subseteq T^*$. Język może być podzbiorem pustym, lecz może także być podzbiorem nieskończonym. Ponadto należy odróżnić język pusty (zbiór $L = \varnothing$) od języka zawierającego jedynie słowo puste (zbiór L = {_})

Przykłady języków to:

L0 = Ø - język „pusty”,

L1 = {_} - język zawierający tylko jedno słowo (puste),

L2 = T* - język zawierający wszystkie słowa, jakie można utworzyć przy użyciu alfabetu T (tzw. język nad alfabetem T),

L3 = {_, 1, 10, 101,1010} - język zawierający skończona liczbę słów,

L4 = {1, 11, 111, 1111, …} - język nieskończony.

Definicja 2.3:

Gramatyką G nazywamy uporządkowaną czwórkę:

$$G = < N, T, P, R >$$

gdzie:

N – zbiór symboli nie terminalnych,

T – zbiór symboli terminalnych,

P – symbol początkowy (czyli także nie terminalny),

R – zbiór produkcji (reguł) , z których każda występuje postaci: $A \rightarrow B$.

przy czym:

$R \subseteq (N \cup T)^* \times (N \cup T)^*$ oraz $R = \{ A \rightarrow B \mid A \in (N \cup T)^*, B \in (N \cup T)^* \}$ ■

Postać reguł wyprowadzania można ograniczać. Otrzymuje się wówczas klasy gramatyk (wprowadzony po raz pierwszy przez Noama Chomsky’ego [Dob92]), które można definiować następująco:

- gramatyki regularne (regular grammars) – czyli gramatyki formalne generujące język regularny (istnieje automat o skończonej liczbie stanów potrafiący zdecydować, czy dane słowo należy do języka). Ten wariant gramatyki formalnej dopuszcza istnienie reguł, które po stronie lewej zawierają dokładnie jeden symbol

nie terminalny, zaś po prawej stronie nie więcej niż jeden symbol nie terminalny i dokładnie jeden symbol terminalny,

- gramatyki bezkontekstowe (context-free grammars) – czyli gramatyki formalne, zawierające reguły wyłącznie w postaci $A \rightarrow B$, gdzie A jest dowolnym symbolem nie terminalnym i jego znaczenie nie zależy od kontekstu w jakim występuje, zaś B to dowolny (także pusty) ciąg symboli terminalnych i nie terminalnych. W praktyce ograniczenie oznacza, iż symbol A nie będzie miał „sąsiadów" stanowiących kontekst, w jakim może być interpretowany,
- gramatyki kontekstowe (context-sensitive grammars) – czyli gramatyki formalne, których reguły posiadają po lewej stronie uzupełnienie (kontekst) dla symbolu nie terminalnego, który tam się znajduje. Postać takiej reguły to $\alpha A \beta \rightarrow \alpha \gamma \beta$, gdzie A jest dowolnym symbolem nie terminalnym, α i β są dowolnymi ciągami symboli terminalnych i nie terminalnych, zaś γ to dowolny niepusty. Tym samym α i β tworzą kontekst dla A w regule.

W przypadku braku ograniczeń dla reguł (tzw. zerowa klasa Noama Chomsky'ego) mamy do czynienia z tzw. gramatyką kombinatoryczną.

Praktyczne zastosowania gramatyk formalnych sprowadzają się głównie do rozpatrywania problemu wyprowadzalności słów języka w zdefiniowanej właśnie gramatyce. Ponieważ symbole alfabetu mogą identyfikować obiekty, a słowa języka mogą mieć znaczenie słownikowe – gramatyki znajdują zastosowania w przetwarzaniu obiektów zlokalizowanych w najróżniejszych dziedzinach. Pojęcie wyprowadzalności danego słowa ω ze słowa λ w gramatyce (G) określamy zapisem:

$$\lambda \Rightarrow_G \omega$$

przy czym: $\lambda = \alpha\gamma\beta$, $\omega = \alpha\gamma\beta$, $\alpha \rightarrow \beta \in R$, $\alpha, \gamma, \beta, \omega, \lambda \in (N \cup T)^*$.

I wówczas słowo ω jest wyprowadzalne ze słowa λ w gramatyce G, jeżeli można ustalić (istnieją):

$$\varphi_0, \varphi_1, \varphi_2, \ldots, \varphi_n \in (N \cup T)^* \text{ takie, że } \varphi_0 = \lambda, \varphi_n = \omega, \varphi_{i-1} \Rightarrow_G \varphi_i \text{ dla } i = 1,2,\ldots,n.$$

Gramatyka jest jednym ze sposobów definiowania języka formalnego. Jak łatwo ustalić, będzie on zbiorem wszystkich słów, które mogą być w tej gramatyce wyprowadzone z jej

symbolu początkowego Z. Zbiór ten (L) nazywamy językiem generowanym przez gramatykę (G) i oznaczamy przez L(G).

Gramatyki kształtu są podstawą funkcjonowania algorytmów realizowanych w ramach wielu rozwiązań wspomagających przetwarzanie danych w grafice. Opisane w kolejnych rozdziałach liczne systemy generacyjne (z gramatykami kształtu na czele) są tego najlepszym dowodem. Analogie pojęciowe do gramatyk formalnych pośród przetwarzanej treści oraz zapożyczenie instytucji reguły jako środka definiującego transformację to znamienne cechy tych systemów.

2.3.2 Gramatyki stochastyczne

W stochastycznym podejściu do gramatyk formalnych został wprowadzony czynnik losowy (probabilistyczny). Każda z reguł gramatyki jest uzupełniana o wartość liczbową wyrażającą prawdopodobieństwo wykorzystania tej reguły – z zachowaniem warunku, iż suma owych wartości dla wszystkich reguł posiadających taką samą lewą stronę wynosi 1. W każdym kroku procesu wyprowadzania słowa wartość czynnika probabilistycznego może ulegać zmianie, jednak zawsze z poszanowaniem powyższego warunku. Często wartość ta jest obniżana dla reguł właśnie wykorzystanych (tym samym rośnie prawdopodobieństwo wykorzystania innych reguł z taką samą lewą stroną w następnych krokach wyprowadzania słowa). Formalnie gramatykę stochastyczną można określić następująco:

Definicja 2.4:

Gramatyką stochastyczną jest czwórka:

$$G = < N, T, P, R_{ST} >$$

gdzie:

N – zbiór symboli nie terminalnych,

T – zbiór symboli terminalnych,

P – symbol początkowy (czyli także nie terminalny),

R_{ST} – zbiór produkcji (reguł) , z których każda występuje postaci: $(A_i \rightarrow B_{ij}, p_{ij})$.

przy czym:

$R_{ST} \subseteq (N \cup T)^* \times (N \cup T)^*$

$R_{ST} = \{ A_i \rightarrow B_{ij} \mid A_i \in (N \cup T)^*, B_{ij} \in (N \cup T)^* \}$ oraz suma $p_{ij} = 1$ dla $i = 1, \ldots, n$

gdzie n to liczba reguł, których lewa strona ma postać A_i. ■

Gramatyki stochastyczne są także teoretycznym punktem wyjścia dla samych gramatyk kształtu. Uznanie reguły za *stosowalną* (rozdział 2.4.1) wobec danego kształtu (materiału graficznego) w gramatyce kształtu wymaga istnienia transformacji generującej tzw. pod-kształt z materiału przetwarzanego. Transformacja, nie posiadając swojej definicji w samej gramatyce kształtu, musi być określana mianem „dopuszczalnej" [Gra07]. Przy materiale nie sklasyfikowanym transformacja jest nierzadko dość dowolnie zadana. Otwiera to problem wyodrębniania pod-kształtu z obrabianego materiału graficznego. Często także wymusza udział czynnika probabilistycznego w transformacjach i stymuluje niedeterminizm przyszłego rozwiązania - zależnie natualnie od środków podjętych przez autorów konkretnego opracowania inżynieryjnego. Z powyższej przyczny teoria gramatyk stochastycznych także dostarcza korzeni dla opracowań metod przetwarzania w systemach generacyjnych. Szersze rozważania na ten temat będą miały miejsce w rozdziale trzecim.

2.4 Gramatyki kształtu

Gramatyki kształtu (*Shape grammars)* wywodzą się z tematyki geometrii analitycznej, w której poprzez zastosowanie zapisu formalnego pojawiła się możliwość definiowania wszelkiego typu obiektów (docelowo także nie-geometrycznych) oraz transformacji, jakim te obiekty mogą podlegać. Gramatyki te przykładem systemów generacyjnych na modelu lingwistycznym. System przetwarzający gramatyką operuje w tak zwanej przestrzeni projektowej wyznaczając zbiór potencjalnych rozwiązań projektowanych, określanych przez elementy tej przestrzeni. Definicja gramatyki kształtu nie narzuca dziedziny w jakiej gramatyka kształtu mogłaby znaleźć zastosowanie. Intuicyjnie przyjmuje się, iż będą to dane reprezentujące kształty w przestrzeni n-wymiarowej. Formalnie określa się elementy słownika gramatyki jako obiekty zlokalizowane w zbiorze tzw. kształtów podstawowych. Teoretycznie elementy te można zastąpić definicjami także zupełnie innych encji, będących przykładowo sygnałem dźwiękowym, elektrycznym, czy obrazem rastrowym. W konsekwencji tego faktu gramatyki kształtu mogą być potencjalnie stosowane także w innych niż grafika dziedzinach.

2.4.1 Formalizm gramatyki kształtu

Gramatyki kształtu zostały po raz pierwszy formalnie opisane w roku 1972 przez Sitny'ego i Gipsa [Sti72]. Opracowane zostały z myślą o wspomaganiu procesu projektowanego z zastosowaniem „lingwistycznego modelu systemu generacyjnego" [Sti75]. Gramatyka kształtu została zdefiniowana w analogii do gramatyki formalnej. Umożliwia określenie procesu generowania wyrażonych graficznie słów języka, zbudowanych z symboli alfabetu za pomocą reguł [Sti80].

Definicja 2.5:

Gramatyka kształtu jest zatem czwórką:

$$G = < K_T, K_N, R, P>$$

gdzie:

- K_T - zbiór kształtów (symboli) terminalnych,
- K_N - alfabet znaczników, czyli innych kształtów (nie terminalnych), przy czym zbiory K_T i K_N są rozłączne,
- P - kształt początkowy (kształt, od którego rozpoczyna się wyprowadzanie słów jezyka gramatyki kształtu). Składowe P należą do K_T i K_N, przy czym przy czym z K_N musi pochodzić przynajmniej jedna składowa,
- R - zbiór reguł kształtu (produkcji). ■

Gramatyka kształtu generuje *język*. W analogii do gramatyki formalnej *język* taki będzie zbiorem kształtów wygenerowanych z kształtu początkowego przy użyciu reguł kształtu (produkcji) gramatyki kształtu. Każdy element języka będzie kształtem terminalnym, ewentualnie częścią takiego kształtu.

W różnych opracowaniach dają się zauważyć pewne niespójności dotyczące terminologii określania elementów gramatyki kształtu. W opozycji do najbardziej znanej definicji W. Mitchela [Mit77] kształt jest także określany mianem „diagramu kształtów". Wówczas nie jest konieczne logiczne dzielenie kształtu na „pod-kształty", z których zbudowane są słowa *języka*.

Każda z *reguł kształtu* składa się z dwóch elementów:

- wyrażenia postawionego z jej lewej strony (tzw. LHS – Left-Hand Side shape), oznaczonego niżej symbolem A,
- wyrażenia postawionego z jej prawej strony (tzw. RHS – Right-Hand Side shape), oznaczonego niżej symbolem B.

Definicja 2.6:

Regułą kształtu określamy zatem wyrażenie:

$$A \rightarrow B,$$

gdzie A i B są kształtami, przy czym A musi zawierać przynajmniej jeden pod-kształt nie terminalny. ■

Reguły można aplikować do przetwarzania kształtu (potocznie: „odpalać" lub „uruchamiać") wówczas, gdy spełnione jest kryterium jej *stosowalności* do danego kształtu.

Definicja 2.7:

Reguła $A \rightarrow B$ jest *stosowalna* do kształtu C wtedy i tyko wtedy gdy możliwe jest wyodrębnienie z A takiego kształtu, który jest pod-kształtem C. Innymi słowy – gdy istnieje *transformacja dopuszczalna* x(A) dająca „pod-kształt kształtu" C. Stosowanie reguły oznacza przeprowadzenie C do nowego kształtu C' poprzez zastąpienie w C kształtu x(A) kształtem x(B). ■

Zastosowanie formalizmu gramatyki kształtu w praktyce oznacza użycie wobec kształtu początkowego kolejnych *reguł stosowalnych* (wybieranych na podstawie osobno zdefiniowanego algorytmu) - aż do momentu wyeliminowania z niego wszystkich pod-kształtów nie terminalnych.

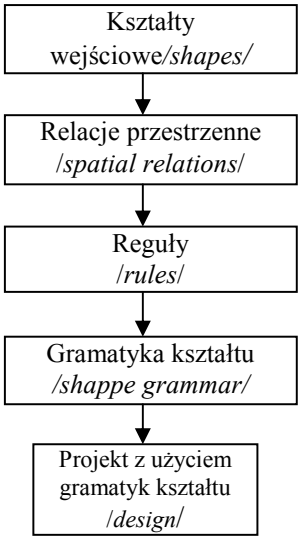
2.4.2 Tworzenie gramatyk kształtu

Gramatyka kształtu określa formalizm, z użyciem którego możliwe jest przeprowadzenie kształtu początkowego w kształty terminalne. Opracowanie treści gramatyki kształtu (sprowadzające się do zdefiniowania zawartości zbiorów tej gramatyki) jest krótko mówiąc procesem dostarczającym odpowiedzi na wymagania funkcjonalne w zakresie konkretnej transformacji kształtów. Projektowany system, wykorzystując gramatykę kształtu, będzie mógł operować zgodnie z tymi wymaganiami. Stiny postuluje [Sti94] uporządkowanie procesu

tworzenia i aplikacji gramatyki kształtu według ustabilizowanego przez siebie schematu. Zakłada on wprowadzenie faz tworzenia gramatyki, przez które proces przetwarzania gramatyki kształtu będzie przechodzić. Fazy te obejmą:

- przetwarzanie kształtów wzorcowych,
- przetwarzanie relacji w przestrzeni pomiędzy kształtami wzorcowymi,
- definiowanie reguł kształtu,
- definiowanie gramatyki kształtu,
- użytkowanie gramatyki kształtu (np. realizacja projektu).

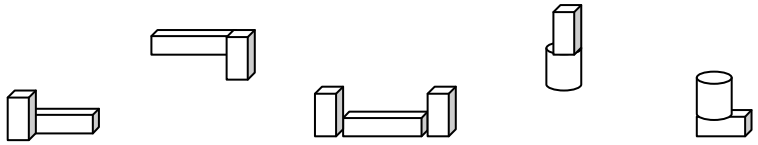
Obrazowo można je przedstawić na diagramie (Rysunek 2.1):



Rysunek 2.1: Cykl tworzenia gramatyki kształtu według prof. Stiny'ego [Sti94].

Metodyka projektowania gramatyki kształtu zakłada istnienie puli rozpoznawalnych kształtów elementarnych /*shapes*/. Termin „rozpoznawalny" oznacza tu tyle, iż są możliwe do stworzenia operatory porównania, jednoznacznie klasyfikujące dane kształty jako zgodne ze wzorcem (będącym kształtem, lub inaczej wyrażonym obiektem) [Sti72]. Dla gramatyki operującej w przestrzeni trójwymiarowej kształtem może być przykładowo równoległościan, stożek, krzywa usytuowana w przestrzeni 3D itp. Po zdefiniowaniu zbioru kształtów (interpretowanych jako alfabet dla danej gramatyki) możliwe staje się ustabilizowanie zbioru relacji wiążących te kształty zwanych relacjami przestrzennymi (*spatial relations*). Relacje te dostarczą informacji o właściwościach topologicznych kształtów określonych nad konkretną

przestrzenią [Cha96]. Wyrażenie relacji przestrzennych wymaga wprowadzenia układu odniesienia (*spatial layout*) określanego w wyniku prowadzenia kalkulacji właściwości topologicznych z wykorzystaniem zdefiniowanej wcześniej arytmetyki [Man96]. Te relacje wypełnią następnie LHS i RHS reguł kształtu. Przykładowa reprezentacja graficzna relacji przestrzennych może wyglądać jak na rysunku 2.2.



Rysunek 2.2: Relacje przestrzenne (*spatial relations*) stosowane w LHS i RHS reguł kształtu (5 przykładów) [Sti82].

Zastosowanie relacji w przestrzennych w regułach kształtu to następny krok. Tutaj, zależnie od przeznaczenia gramatyki kształtu po stronie lewej (LHS) znajdują się albo wyłącznie kształty (gramatyki wyłącznie syntezujące kształty rozbudowane z elementarnych), albo także relacje przestrzenne (w przypadku gramatyk bardziej zaawansowanych, czyli będących w stanie również przetwarzać kształty złożone. Przykładowo, w przypadku gramatyki syntezującej, pula kształtów (*shapes*) może być oznakowana symbolami A,B,C,D,E. Wówczas przykładowe relacje przestrzenne to A+C, A+D+E, C+E, B + C itp. Należy pamiętać, iż operator „+" nie oznacza w tym przypadku jedynie dowolnej unii dwóch kształtów – przenosi on bowiem dodatkową informację o ułożeniu tych kształtów względem siebie. Może zatem istnieć jednocześnie kilka przypadków związku kształtów (np. kształtu A i kształtu B), określanych tym samym wyrażeniem: A+B. Tutaj postuluje się często wprowadzenie dodatkowych oznaczeń symbolicznych, rozróżniających je. Wychodzimy zatem poza klasyczne gramatyki kształtu, dodając do LHS i RHS dodatkowe symbole. O ile w przypadku RHS reguły kształtu mogą mieć przykładowo:

$$A \rightarrow A + C$$

$$C \rightarrow C + E$$

$$B \rightarrow A + D + E$$

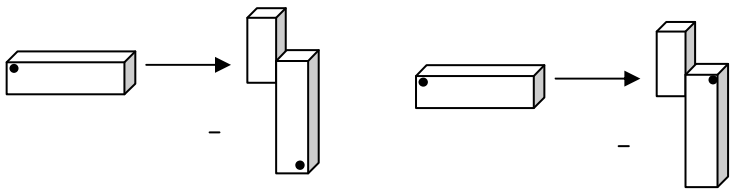
i dalej być zgodne z definicją gramatyki kształtu, o tyle w przypadku LHS dodatkowe symbole wprowadzą konieczność istnienia kształtu towarzyszącego warunkującego stosowalność reguły

i tym samym – kontekstowość gramatyki. Tym samym w przypadku gramatyk zawierających takie reguły kształtu wyrażenie dodatkowo zawarte w LHS będzie stanowić tzw. *kontekst*, w którym stosowalność reguły istnieje:

$$A + C \rightarrow A + D + E$$

$$B + C \rightarrow A + C$$

Problemem rozwiązywanym dodatkowo na etapie definiowania reguł kształtu jest rozpoznanie orientacji obiektu w przestrzeni. Każda bryła posiadająca w przestrzeni 3D przechodzącą przez jej środek płaszczyznę asymetrii wymaga takiej orientacji. Prawidłowa orientacja kształtu jest konieczna w późniejszych procesach dopasowywania reguł oraz procesach uruchamianych w momencie „odpalenia" reguły – czyli dokonujących faktycznych zmian w obrabianym projekcie. Najprostszym sposobem przeniesienia informacji o orientacji kształtu jest wyróżnienie jednego z jej punktów (np. „narożnika" bryły) i zapisanie informacji o tym wyróżnieniu w regule kształtu (czynność tą w literaturze określa się przeważnie mianem *labelingu reguł kształtu i kształtów*) [Mus10]. Tak samo zadana reguła, np. $A \rightarrow A + C$ może w efekcie zastosowania produkować różny wynik. Użycie *labelingu* ilustruje rysunek 2.3.



Rysunek 2.3: Przykład różnego *labelingu reguły kształtu oznakowanej* jako $A \rightarrow A + C$, dający różne efekty wynikowe [Mus10].

Zdefiniowanie zbioru reguł kształtu otwiera drogę do opracowania strategii ich stosowania. Jest to najtrudniejszy etap tworzenia rozszerzeń gramatyki. Sam operator porównania kształtów nie daje gwarancji deterministycznego funkcjonowania przekształceń z prowadzonych użyciem gramatyki. Bardzo łatwo można wyobrazić sobie sytuację, kiedy dopasowano dwie reguły i powstaje problem – która z nich powinna być użyta. Temu zagadnieniu poświęcono wiele opracowań, ograniczając zwykle finalne rozwiązania do wąskiej puli możliwych kształtów, przy których proponowana gramatyka daje pewność ich deterministycznego przetworzenia. Tworzono także opracowania angażujące wiele heterogenicznych gramatyk jednoczesnie, lecz także w wąskich dziedzinach zastosowania (np. *discursive grammars* Jose'a Duarte [Dua00]. W przypadku zastosowań gramatyk kształtu do

graficznego wspomagania projektowania jakiegokolwiek modułu decyzyjnego wyrokującego o uruchamianiu reguł często w ogóle się nie implementuje – pozostawiając każdorazowo decyzję użytkownikowi lub uzależniając decyzję od precyzyjnych diagramów sterujących [Gra07]. To ostatnie rozwiązanie często znacznie redukuje wartość użytkową metody przetwarzania - ograniczając je do sekwencyjnie wykonywanej procedury składającej się z serii zamian elementów (pod-kształtów) graficznych na inne.

Przez ponad trzydzieści lat koncepcja gramatyk kształtu była wyjściem dla szeregu różnorodnych opracowań, mających na celu udoskonalenia lub adaptację skromnego modelu przetwarzania bazującego na regułach kształtu do zastosowań praktycznych w grafice. Z czasem pulę zastosowań gramatyk kształtu znacznie rozszerzono obejmując rozwiązaniami przetwarzanie wszelkich sygnałów, które tylko można poddać skutecznej identyfikacji (w całości lub fragmentarycznie). Obok kształtów geometrycznych pojawiły się ich atrybuty rozszerzone (jak kolor, masa czy materiał). Potem przyszła kolej na obróbkę zapisów symbolicznych, kształtów zadanych funkcyjnie (np. krzywe), a nawet materiałów multimedialnych (np. dźwięk) czy przebiegów czasowych sygnałów elektrycznych.

2.4.3 Przetwarzanie regułami kształtu

Praktyczne zastosowania *gramatyk kształtu*, które ograniczają rozważania do niewielkiej puli pre-definiowanych kształtów nie przedstawiają wielkiej wartości użytkowej. Zdefiniowane *transformacje* wyodrębniające kształt muszą operować na takiej jego notacji, aby operacja wyodrębniania i następnie rozpoznawania go była możliwa do przeprowadzenia. W modelu abstrakcyjnym, na jakim bez wątpienia umieszczone są gramatyki kształtów znaczniki i kształty terminalne są jednoznacznie rozpoznawalne. Materiał pochodzący ze środowiska systemu wykorzystującego *gramatyki kształtów* do przetwarzania tej cechy posiadał nie będzie. Zatem kluczowym problemem okazuje się stworzenie mechanizmu klasyfikacji obiektów - będących przyszłymi kształtami w tworzonej gramatyce. Jak wiadomo - reguły kształtu mogą także modyfikować analizowany kształt (aplikując transformacje). Produkt tej transformacji (a konkretnie jego reprezentacja graficzna) także musi być rozpoznawalny.

Innym problemem jest tworzenie mechanizmów sterujących kolejnością uruchamiania reguł. Tutaj najpopularniejszą metodą jest wprowadzenie diagramu sterującego, określającego sekwencję kolejno uruchamianych reguł kształtu. Zagadnienie to jest jak najbardziej ciekawe i będzie przedmiotem rozważań w niniejszej pracy.

Jak wspomniano w rozdziale 2.4.1, *regułę kształtu* można określić wyrażeniem: $A \rightarrow B$, gdzie A i B są kształtami, przy czym A musi zawierać przynajmniej jeden pod-kształt nie terminalny. Ustalono też, że *stosowalność* reguły do kształtu C osiągamy wówczas, gdy istnieje transformacja dopuszczalna t(A), za pomocą której możliwe jest odseparowanie pod-kształtu C. W praktyce określenie reguły jako *stosowalnej* do przetworzenia danego kształtu oznacza rozwiązanie problemu tzw. dopasowania reguły i danego kształtu. Reguła kształtu może potencjalnie przeprowadzać kształt C dopasowany do A w kształt C' będący transformacją kształtu C zadaną treścią reguły kształtu (produkcji). Dopasowanie reguły nastąpi wówczas, gdy kształt A będzie odpowiadał fragmentowi kształtu C (fragment ten oznaczmy jako D) w myśl definicji *stosowalności reguły kształtu*.

Tutaj następuje nawrót do problemu wskazanego w podrozdziale 2.4.1, gdyż zdefiniowanie transformacji separującej kształt, a w konsekwencji i metryki jednoznaczne wyrokującej o „dopasowaniu kształtu A do C" nie jest zagadnieniem banalnym. W przypadku klasycznym zakłada się przykładowo, iż reprezentacja graficzna w przestrzeni metrycznej Euklidesa kształtu A musi być identyczna z reprezentacją C. Transformacje te na etapie dopasowywania przeprowadzają fragment A w nowy kształt, posiadający gabaryty oraz usytuowanie identyczne jak kształt będący wzorcem. Jeśli po takich transformacjach został osiągnięty kształt identyczny – nastąpi dopasowanie. Proces dopasowywania obciążony jest licznymi wadami, związanymi z niedoskonałościami metryk dopasowania czy znacznym uproszczeniem *transformacji dopuszczalnej*.

Problem stosowalności reguł kształtu ma też wpływ na określenie warunku stopu dla algorytmu przetwarzającą kształty gramatyką kształtu. W niektórych realizacjach zakłada się istnienie jednej, wyróżnionej reguły kształtu (terminatora) [Mit78]. Odpalenie tej reguły było jednoznaczne z zakończeniem procesu – czyli usunięciem ostatniego pod-kształtu nie terminalnego.

Podsumowując – opracowanie dobrego algorytmu sterującego aplikowaniem reguł, według których przetwarzany będzie materiał graficzny wymagać będzie szeroko zakrojonych rozważań. Należy je poprzedzić dokładną analizą rozwiązań pokrewnych do teorii gramatyk kształtu, co umożliwi poszerzenie poglądu na możliwości ewentualnej rozbudowy mechanizmu sterującego wykorzystującego reguł. Dzięki temu systemy tworzone w oparciu o taki mechanizm mogłyby udowodnić istnienie możliwości rozwiązania problemów, na które zwrócono uwagę w pierwszym rozdziale pracy.

3 Istniejące podejścia do problemu przetwarzania kształtów

Obecny rozdział omówi szerzej problemy stawiane przed rozwiązaniami przetwarzającymi bardziej lub mniej formalnie zdefiniowaną wiedzę na potrzeby zastosowań graficznych. Określi także zakres prac, które w ostatnich latach były realizowane w ramach tej tematyki. Przejdzie także do rozważań nad elementami metodyki wykorzystywania graficznych systemów generacyjnych, zalet i wad takich systemów, możliwości ich rozwijania i wprowadzonych już ulepszeń. Idąc tą ścieżką uwypukli najbardziej istotne problemy podjęte w opracowaniach obecnie dostarczonych rozwiązań i skonfrontuje je z ogólnie przyjętymi celami niniejszej pracy – otwierając drogę do przyjęcia założeń dla proponowanych w pracy rozwiązań.

3.1 Rozwiązania oparte o graficzne systemy generacyjne i modele pokrewne

Wiedza oparta na regułach była już wielokrotnie adaptowana jako nośnik informacji umożliwiający wspieranie przetwarzania obiektów graficznych w różnorodnych zastosowaniach. Projektowanie systemu przetwarzającego wiedzę na potrzeby zastosowań graficznych spotyka się z podobnymi problemami jak te, dotyczące systemów bazujących ogólnie na regułach. Projekty konkretnych systemów mających na celu przetwarzanie bazy wiedzy zawierającej reguły (czyli zwanych potocznie systemami regułowymi) są przeważnie odpowiedziami na dość uniwersalne problemy związane z samym przetwarzaniem wiedzy. Owe typowe zagadnienia koncentrują się wokół tematyki określającej:

- pozyskiwanie wiedzy *(knowlegde acquisition)* będącego procesem pobierania wiedzy od eksperta w dziedzinie. Ekspert w dziedzinie musi mieć możliwość przekazania posiadanej wiedzy. Postać tej wiedzy jest abstrakcyjna i może być ukonkretniana inaczej w każdym przypadku. Opracowanie jak najbardziej uniwersalnych mechanizmów jej pobierania leży u podstaw problemu,

- kodowanie (reprezentację) wiedzy (*knowledge representatnion*) definiujące techniki utrwalenia pozyskanej wiedzy. Pobrana wiedza musi być utrwalona przy użyciu struktur, których treść będzie zależeć od postaci wiedzy pobranej od eksperta oraz od dalszych wymagań stawianych systemowi. Stworzenie struktury optymalnej jest celem zagadnienia związanego z kodowaniem wiedzy,
- kontrolę procesu przetwarzania wiedzy (*reasoning control*) określającą procedury przetwarzania i interpretacji wiedzy. Procedury te mogą być także wyrażone samą wiedzą (zwaną meta-wiedzą). W takim przypadku od treści samej wiedzy może zależeć dalsze wykorzystanie reguł,
- weryfikację poprawności wiedzy (*knowledge verification*) zakładającą rozwiązanie problemu kontroli treści wiedzy pod kątem kryteriów poprawności określonych przez ustaloną reprezentację wiedzy,
- prezentację rozwiązań (*explaining solutions*) wyrażającą problem polegający na utrzymaniu możliwości zaprezentowania wyników ekspertyz z użyciem wiedzy.

W przypadku gramatyki kształtu, również angażującej wiedzę opartą na regułach, pula wtórnych problemów do rozwiązania ma swoje źródło w konkretnych zapotrzebowaniach inżynieryjnych. Na przestrzeni ostatnich kilkunastu lat w problematyce tej przewijały się przykładowo następujące zagadnienia [Ger04], [Gip75]:

- umożliwienie automatycznego rozpoznawania kształtów zawartych w przestrzeni Euklidesa,
- wprowadzenie parametrycznych reguł kształtu,
- umożliwienie automatycznego rozpoznawania kształtów parametryzowanych,
- budowa gramatyk przetwarzających kształty trójwymiarowe,
- rozpoznawanie i przetwarzanie kształtów zadanych funkcyjnie (np. krzywe),
- rozwijanie gramatyk wyodrębniających pod-kształty,
- rozbudowa modułów wnioskujących - głównie w celu eliminowania nie deterministycznych procesów przetwarzających kształty z użyciem gramatyki kształtu.

Jak widać – liczba dróg postępowania przy wprowadzaniu innowacji jest dość spora. Większość zespołów badawczych, wychodząc naprzeciw konkretnym zapotrzebowaniom inżynieryjnym, starała się rozwiązać wybrane z powyższych problemów poprzez tworzenie interdyscyplinarnych rozwiązań łączących gramatyki kształtu z osiągnięciami w innych dziedzinach. Większość udokumentowanych publikacjami udanych eksperymentów polega na wykorzystaniu założeń gramatyk kształtu przy budowie rozmaitych transformacji dokonujących modyfikacji szeroko pojętego sygnału wejściowego – nawet zadanego nie tylko klasycznie rozumianymi geometrycznymi "kształtami". W bieżącym podrozdziale dokonany zostanie przegląd ciekawszych zastosowań gramatyk kształtu wraz z omówieniem osiągniętych przez poszczególne rozwiązania wyników. Stanowić to będzie ważny materiał dla szacowania celowości podejmowania określonych kierunków badań przy stabilizowaniu koncepcji Kaskadowych gramatyk kształtu (K-GK) opisanej w rozdziale piątym.

3.1.1 Próby automatyzowania procesów projektowania grafiki z użyciem gramatyk kształtu

Sztandarowym przykładem zastosowania gramatyk kształtu jest wspomaganie systemów CAD/CAM. Mnogość opracowań na ten temat sugeruje, aby wstępie poświęcić temu zagadnieniu nieco uwagi. Budowa tzw. interpreterów kształtów, mających za zadanie ocenić możliwość zastosowania danej reguły kształtu jest tu dość prosta. Sprowadza się do unifikacji sposobu kodowania obrabianych kształtów pomiędzy regułami i formatem danych, na którym operuje narzędzie wspomagane. Unifikacja taka powoduje, iż reguły operują na formacie danych zgodnym z wewnętrznym formatem, stosowanym na przykład przez edytor graficzny czy inne tego typu narzędzie. Interpreter kształtu redukuje wówczas swoją funkcjonalnośc do przetwarzania operatorów porównania tych zdentyfikowanych już danych. W bardziej zaawansowanych przypadkach zaistnieje konieczność dokonywania dodatkowych przekształceń – z reguły bezpośrednio przed zastosowaniem operatora porównania (np. poszukiwanie pasującego wycinka danych, skalowanie itp.). Szukanie wycinka kształtu, często rozważane w literaturze ([Lia05], [Fle87]) i określane mianem poszukiwania *relacji części* (*part relation*) jest problemem szczególnie złożonym. Poświęcono mu także pewną część niniejszego opracowania.

Unifikacja jest łatwa do przeprowadzenia jeśli całość sygnału wejściowego została wyprodukowana z pomocą dobrze znanych funkcji i od razu pod postacią korzystnego z punktu

widzenia operatorów porównania formatu. Tak będzie się działo, gdy pochodzi ona w całości np. z edytora graficznego.

Moduł kontroli uruchamiania reguł jest tu często całkowicie zredukowany, pozostawiając decyzję o odpaleniu reguły użytkownikowi. Jedynym problemem jest dostarczenie mu odpowiedniego interfejsu, umożliwiającego podejmowanie takich decyzji. Propozycję takiego interfejsu można zapisać formalnie, przykładowo podobnie jak zrobił to Andrew I-Kang Li [Lia05]. Rozważa ona fazę projektu D = <C, d>, gdzie C jest kształtem, a d jego opisem. Każda następna faza projektu typu <C(i + 1), d(i + 1)> jest wywiedziona z poprzedniej (<Ci, di>) przykładowo według formuły:

```
if g(t(A)) ≤ Ci, then C(i + 1) = [Ci – g(t(A))] + g(t(B))
```

gdzie A i B to LHS i RHS reguły A → B. Metryka, określająca zgodność reguły z wymaganiami użytkownika interfejsu zadana jest funkcją g(). Umożliwia ona dodatkowe parametryzowanie operacji t(), zadanej przez użytkownika. Operator ≤ dostarczy jako wynik informację o dostatecznej zbieżności wartości zwracanej przez g() z kształtem C (w tym przypadku C1). Korzystając z takiego operatora interfejs może zachowywać się następująco:

1. przedstawia bieżącą fazę projektu,
2. informuje, jakie reguły mogą być zastosowane do <Ci, di> w ramach operacji t() wybranej przez użytkownika, która jest dodatkowo doprecyzowana wartościami parametrów, co obrazuje funkcja g(),
3. na żądanie użytkownika przedstawia możliwy wynik - po zaaplikowaniu rozważanej przez użytkownika reguły, prezentując $C_{(i+1)}$ obliczone z użyciem wzoru:
   ```
   C(i + 1) = [Ci – g(t(A))] + g(t(B))
   ```
4. pobiera od użytkownika decyzję o wyborze reguły,
5. stosuje regułę, przeprowadzając projekt w kolejną fazę.

Przy złożonych procesach transformacji kształtów nie tylko w tym rozwiązaniu daje się zauważyć tendencję do uzależniania przebiegu faz procesu od czynnika zewnętrznego (którego dostarczenie jest konieczne). W przypadku technicznej możliwości prezentacji wyników pośrednich czynnik ten jest dostarczany często w sposób najbardziej prozaiczny - przez użytkownika.

3.1.2 Wprowadzanie arytmetyki kształtów

Pojęcia „arytmetyki kształtów" dość często występuję w publikacjach rozwijających teorie automatycznego przetwarzania kształtów. Każdorazowo konieczne jest tu zaangażowanie systemu walidacji kształtów, umożliwiającego ich klasyfikację lub rozpoznanie. Autorzy często nie przedstawiają żadnej formalnej definicji „arytmetyki kształtów", opisując jedynie konsekwencje jej wprowadzenia, niejednokrotnie wyłączenie w poprzez przedłożenie w pracy konkretnych przykładów obrabianego materiału graficznego.

Zdefiniowanie samej arytmetyki kształtów niewątpliwie wymaga wprowadzenia spójnego systemu ich klasyfikacji, a zaraz po nim – ustanowienia zbioru relacji pomiędzy nowopowstałymi klasami obiektów, wchodzących w skład przetwarzanego materiału graficznego. Poza relacjami (z których proste to przykładowo zawieranie lub równoznaczność) arytmetyka kształtów wprowadza także operatory, których specyfikacja często jest otwarta (poza prostymi operatorami typu *substraction* /-/, *addition* /+/ wprowadza się abstrakcyjnie ujęte transformacje (*shape transformations*). Z tego właśnie powodu na bazie arytmetyki kształtu było możliwe wyprowadzenie teorii gramatyk kształtu, o których szersze rozważania będą miały miejsce w następnych podrozdziałach.

3.1.3 Próby formalnej walidacji kształtów

Inna gałąź zagadnień, związanych z przetwarzaniem wiedzy o kształtach 2D i 3D jest walidacja modeli 3D osiągniętych jako wyniki. Celem opracowań jest wówczas stworzenie bazującego na regułach systemu dowodzenia poprawności dostarczonego kształtu (*rule based geometry prover*).

Systemy, powstałe w drodze rozwijania takich zagadnień, są przeważnie silnie uzależnione od dziedziny kształtów graficznych, do obróbki których zostały zaprojektowane. Przykładowo mogą operować na zestawie prymitywów (na przykład linii prostych) zadanych w dwuwymiarowej przestrzeni Euklidesa. Kształt początkowy jest wtedy materiałem otrzymywanym w wyniku funkcjonowania innych narzędzi. Procedura przetważania takiego materiału będzie wtedy następujaca:

- identyfikacji i opisaniu komponentów kształtu (przypisanych do klas),

- sekwencyjnym przeglądaniu bazy wiedzy (reguł) i sprtawdzaniu tymi regułami komponentów kształtu,
- w konsekwencji przetworzenia – wykazaniu sprzeczności i/lub nieniesieniu korekt w przetwarzanym materiale.

Naturalnie pula relacji opisujących konkretne komponenty testowanego kształtu jest tu opracowywana każdorazowo – w zależności od dziedziny problemów, w jakich metoda ma zastosowanie. Przykładem opracowania rozwijającego wspomniane zagadnienie może być [Cho06] system dowodzący poprawności twierdzeń o kątach zawartych pomiędzy prostymi w przestrzeni 2D. Zestaw reguł opracowano tu na bazie definicji operatorów interpretujących występowanie wzajemnych relacji pomiędzy poszczególnymi komponentami kształtu. Przykładowo: coll (collinear), para (parallel), perp (perpendicular), cong (congruent), eqangle (equalangle). Dodatkowo (przykład silnego uzależnienia od dziedziny kształtów) zdefiniowanego symboliczną notację, umożliwiającą identyfikację terminu kąta pomiędzy liniami prostymi i posłużenie się nim jako operandem: `›[CD, AB]` oznaczało kąt pomiędzy odcinkami definiowanymi przez punkty C, D, A i B. Naturalnie definicji tej towarzyszyła analogiczna definicja linii prostej nieskończonej oraz odcinka. Po wprowadzeniu operatorów i operandów możliwe było definiowanie reguł. Przykładowo:

```
R1:  › [AB, CD] = › [CD, AB]
R2:  If G is on line NM, then › [AC, NM] = ›[AB, PQ]
R3:  If AB is parallel to CD , then ›[AB, CD] = › [FG, NM]
```

gdzie wszystkie symbole literowe oznaczają wierzchołki

Powyższy zbiór jest naturalnie absolutnie elementarny i w praktycznych zastosowaniach jest rozszerzany o kolejne reguły, angażujące kolejne kształty i dalsze relacje (np. wieloargumentowe) typu `circumcenter(Q, A, C, E, D)`, `midpoint(I, E, F)`, `coll(D, I, A)` itp.

3.1.4 Funktory kształtów jako rozszerzenie arytmetyki kształtów

Jednym z bardziej popularnych zabiegów, mających na celu wprowadzenie udogodnień w aplikowaniu gramatyk kształtu do modyfikacji nieokreślonych początkowo obiektów jest użycie konstrukcji, które w języku polskim można okreslić mianem *funktorów kształtów* [Dyd78]. Funktor taki partycypuje w regule, umożliwiając zdefiniowanie w niej bardziej

zaawansowanych przekształceń niż sama substytucja pod-kształtów. Użycie funktorów wymaga zdefiniowania operacji algebraicznych na kształtach, umożliwiających przykładowo dodawanie kształtu do elementów zawartych już w obrabianej przestrzeni, odejmowanie, obliczanie części wspólnej kształtów itp. W większości przypadków takie operacje definiowane są analogicznie do operacji boolowskich w przestrzeni 3D – z zastosowaniem funktorów dwu, a czasem jedno argumentowych. Przykładowo [Yvo03]:

- operacja dodawania 3D brył A i B będzie produkowała będzie produkować bryłę zawierającą w swojej objętości wszystkie punkty, które należą do A lub do B,
- operacja odejmowania 3D brył A i B będzie produkowała bryłę zawierającą w swojej objętości wszystkie punkty, które należą do A i nie należą do B,
- operacja mnożenia 3D brył A i B będzie produkowała bryłę, zawierającą w swojej objętości wszystkie punkty należące do A i jednocześnie należące do B.

Przy takich funktorach zapis LHS i RHS reguł kształtu może przybierać postacie:

$$A_1 + A_2 \Rightarrow A_1 \text{ OR } A_2$$

$$A_1 + A_2 \Rightarrow A_1 \text{ AND } A_2 \text{ MUL } A_1$$

Tutaj naturalnie osobną kwestią stanie się zdefiniowanie metody realizującej takie operacje – czyli tworzącej nowe bryły zgodnie z założeniami funktora. [Yvo03] zawiera propozycję metody nazwanej przez autorów metodą sekcji (*section method*), formalizując booleanowskie operacje trójwymiarowe. Zakłada się tu opisywanie tych operacji wielokrotnymi rzutami operandów (czyli obiektów 3D) na płaszczyzny 2D. Rzut taki zawiera reprezentację graficzną operandów. Określa także wynik operacji.

Warto rozważyć, że definicja funktora kształtu może być rozbudowywana do postaci uwzględniającej parametry środowiskowe procesu przetwarzania konkretnego kształtu. Tym samym uzależnić go można od cech innych kształtów występujących w LHS reguł i traktowanych jako parametry dla procesu przetwarzania kształtu.

3.1.5 Gramatyki kształtów w rozpoznawaniu obrazów

Istnieje sporo przykładów wprowadzania półformalnego przetwarzania materiału graficznego pochodzącego bezpośrednio ze środowiska systemu przetwarzającego. Kształt jest wtedy postrzegany jako sygnał, który przed zbudowaniem z niego kształtu początkowego musi przejść cały szereg transformacji. Jako jedno z ciekawszych opracowań można tu przytoczyć choćby pracę [Mor05], gdzie gramatyki kształtów dostarczyły rozwiązanie biometrycznego problemu analizy liści roślin, bazującej głównie na klasyfikacji ich cech naturalnych – w tym przypadku wygląd krawędzi liści. W tym przykładzie została trochę inna metodyka. Dokonywana jest dekompozycja obiektu wejściowego (liścia) na część główną (centralną) i obrzeża. Cześć centralna nie może być dalej dekomponowana (nie zwróci już żadnych istotnych informacji). Służy jednak (po aproksymacji do prostych figur geometrycznych) jako podstawa dla rozpoznania usytuowania i kształtu krawędzi. Tym samym zastosowanie formalnie przetwarzanych reguł ma na celu klasyfikację kształtów poprzez ich dekompozycję i określanie obecności poszczególnych komponentów.

3.1.6 pL-Systemy dla kształtów

Kolejnym wartym uwagi podejściem jest rozbudowa systemu klasyfikacji kształtów graficznych w oparciu o wykorzystanie systemów Lindemayera, ogólnie znanych pod mianem L-systemów [Roz80]. System taki posiada gramatykę G definiowaną w dość prosty sposób:

$$G = (V, S, \omega, P)$$

gdzie:

V jest zbiorem symboli identyfikujących elementy, mogące ulec wymianie (*variables*),

S jest zbiorem symboli identyfikujących elementy stałe (constants),

ω jest symbolem startowym (initiator),

P jest zbiorem reguł (production rules).

Kluczową dla wykorzystania w grafice cechą tych systemów jest to, że wiele reguł może być stosowanych jednocześnie (równolegle) podczas iteracji. Stąd systemy te często określane są mianem pL-Systemów. pL-Systemy (czyli pararell L-Systemy) są w grafice komputerowej powszechnie znane głównie z zastosowań przy rekursywnym generowaniu kształtów samo-

podobnych (fraktali). Reguły w LHS zawierają najczęściej tylko pojedyncze symbole - powodując tym samym, iż gramatyka jest bez kontekstowa (context-free). Możliwe jest jednak wprowadzenie reguł, zawierających w LHS nie tylko pojedyncze symbole ale także ich sąsiadów – wówczas będziemy oczywiście mieli do czynienia z gramatyką kontekstową (context-sensivive).

Rozszerzenia pL-Systemów w zastosowaniach graficznych polegają często [Doo78] na wprowadzeniu do reguł symboliki reprezentującej wierzchołki lub wręcz wyróżnione fragmenty trójwymiarowej siatki wielokątów przetwarzanej gramatyką. LHS każdej z reguł nie będzie wówczas liniową sekwencją symboli, lecz uporządkowanym zbiorem fragmentów przetwarzanej siatki. Struktura topologiczna siatki może być wówczas porządkowana, a co ważniejsze – modyfikowana zgodnie z porządkiem wyrażonym za pośrednictwem reguł. Możliwe jest wówczas typowanie fragmentów siatki, podlegających ściśle określonym procesom. Kryteria typowania można także rozszerzać na obszary sąsiednie, lub uzależniać klasyfikację od sąsiadów (korzystając z kontekstowości gramatyki). Przykładem zastosowania takiego podejścia jest choćby zaproponowana przez Stefana Maierhofera [Mai02] transformacja umożliwiająca „wygładzanie" siatek wielokątów algorytmami podziału wielokątów (*mesh-subdivision*). Operując wyłącznie na obiektach bezpośrednio tożsamych z komponentami obrabianej trójwymiarowej siatki wielokątów oraz wiedzy określającej relacje pomiędzy nimi (utrzymywanej treścią reguł) nie ma konieczności wprowadzania oddzielnej symboliki, na której operują reguły. Tą symbolikę już utworzyły rozróżnialne komponenty siatki opisującej przetwarzany kształt.

3.1.7 Gramatyk dzielące (*split grammars*)

Tytułowy termin określa kolejną modyfikację gramatyk kształtu. Jest ona o tyle istotna, iż częściowo nawiązuje do rozważań poruszanych w dalszych rozdziałach niniejszej pracy. Zakłada zredukowanie obszaru zastosowań gramatyki kształtu jedynie do wąskiej dziedziny operacji wykonywanych podczas uruchamiania reguł gramatyki. W tym konkretnym przypadku chodziło o operacje podziału istniejącego kształtu na komponenty (kształty proste). Po raz pierwszy zaprezentowana została przez pewien zespół zajmujący się zastosowaniami architektonicznymi dla gramatyk kształtu [Won03]. Konkretne opracowanie inżynieryjne na bazie tego rozwiązania określono mianem „*Instant architecture*" . Zauważono, iż dekompozycja gotowych obiektów architektonicznych do kształtów prostych może zostać zautomatyzowana w

taki sposób, iż powstały produkt (kształty proste) będzie mógł zostać użyty ponownie przy tworzeniu z niego nowych obiektów architektonicznych. Przy takim założeniu opracowana została gramatyka. Zbiór symboli terminalnych jest w niej odpowiednikiem zbioru tych symboli, które nie mogą być dalej dekomponowane (kształtów prostych, stanowiących definiowane gramatyką komponenty architektoniczne). Sam „podział" jest określany jako dekompozycja kształtu bazowego na inne kształty ze słownika B. W takiej gramatyce dzieląca (*split grammar*) występują dwa typy reguł:

- reguła podziału (*split rule*) to reguła typu a → b, gdzie a jest ciągiem symboli stanowiących podzbiór B, zaś b stanowi kolejny ciąg – różniący się w oczywisty sposób treścią. Warto tu zauważyć, iż wprowadzenie pojęcia „ciąg symboli" wymusza kontekstowość gramatyki,
- reguła konwersji (*conversion rule*) to reguła także typu a → b, gdzie a i b są analogicznie ciągami symboli stanowiących podzbiory B. Reguła umożliwia podmianę symbolu bez dekompozycji.

Metryka dopasowania LHS jest oparta o wyrażone skalarnie przedziały wartości, charakteryzujące poszczególne kształty, a tożsame przykładowo z kolorem obiektu reprezentowanego przez kształt, lub typem materiału z którego docelowo jest on wykonany. Tym samym reguły kształtu mogły dodatkowo interpretować wartości powiązane z symbolami w zbiorze B. Taka ścieżka postępowania znzlazłą swoją kontynuację w kolejnych opracowaniach [Kor08].

3.1.8 Gramatyki kolorowane (*color grammars*)

Mianem gramatyk kolorowanych (*color grammars)* określa się kolejny zbiór rozwiązań bazujących na gramatykach kształtu. Tym razem polegają one na użytkowaniu kształtów o poszerzonej puli atrybutów charakteryzujących je [Shg07]. Ma to na celu ułatwienie identyfikacji kształtów i tym samy dopasowanie do LHS reguł. Pierwsze opracowania uregulowały samą genezę nazwy tej modyfikacji. Polegały one faktycznie na wprowadzeniu koloryzacji kształtów. Metodologię tworzenia *color grammars* uporządkował w 1989 roku prof. Terry Knight [Kni89]. *Color grammars* rozwijały się często dość spontanicznie, głównie za sprawą gotowych opracowań inżynieryjnych dedykowanych do konkretnych celów (najczęściej były to znów zastosowania architektoniczne) [Hei94]. Angażowały często różnorodne atrybuty

kształtów (materiał służyć do budowy obiektu reprezentowanego przez kształt, jego przeznaczenie czy cechy wywodząc się z bezpośrednio dziedziny zastosowań) [Kni99].

3.1.9 Gramatyki sortujące (*sortal grammars*)

Próbę wymuszenia zróżnicowania kształtów przetwarzanych przez metody formalne wraz z interpretacją ich logicznej przynależności do kształtów kolejnych podjął Rudi Stuoffs [Stu07]. W odrębnych opracowaniach Stuoffs postuluje wykorzystanie relacji typu „całość-część" pomiędzy kształtami przetwarzanymi w procesie porządkowania ich struktury. Każdy kształt posiadać może pod-kształty, które też są kształtami. Dekompozycja przebiega zdobnie z porządkiem sąsiedztwa kształtów. Z kształtu mogą zostać wyodrębnione kolejne, zajmujące pozycje na jego krawędziach. W przypadku przestrzeni dwuwymiarowej będzie to sąsiedztwo w układzie „północ-południe-wschód-zachód". Zdekomponowane kształty są zatem porządkowane w uporządkowaną strukturę posiadająca skończoną i łatwą do obliczenia ilość komponentów. Podobne postępowanie stosowane jest przy alokacji kształtów w przestrzeni trójwymiarowej. Algebra definiowana na podstawie takich relacji może być rozwijana na wiele sposobów, a jej wyrażenia mogą określać operacje typu: suma, różnica czy w końcu same transformacje kształtów [Stu97]. Takie uporządkowanie kształtów umożliwiło w konsekwencji także opracowanie wydajnych procedur wyszukiwania kształtów celem ich dopasowania [Kri92]. Ważne dla konkretnych opracowań inżynieryjnych jest to, że kształty będące komponentami nie musza mieć natury fizycznej (nie muszą być bryłami posiadającymi reprezentacje fizyczną). Takim komponentem może być atrybut kształtu, na przykład materiał, czy rodzaj reprezentacji graficznej. W komponentach kształtu można wtedy lokalizować wszelkie aspekty wymagające uregulowania przy budowie bądź projektowaniu obiektu, króry jest reprezentowany przez kształt nadrzędny.

3.1.10 Gramatyki grafowe (*graph grammars*)

Dość znaczącym rodzajem systemu generacyjnego, pokrewnym z rozwiązaniami prezentowanymi w pracy są gramatyki grafowe (*graph grammars*) [Cun94]. W literaturze występują one czasami także pod nazwą *gramatyk grafów generacyjnych* [Gra07]. Materiał przetwarzany przez takie gramatyki jest uporządkowany w strukturę grafową. Gramatyka grafowa zawiera reguły operujące na fragmentach grafów. Reguły określają, który fragment grafu można zamienić na inny. Proces tworzenia słowa w takiej gramatyce polega na

rozpoznawaniu podgrafów, a następnie na dokonywaniu wymian tych podgrafów zgodnie z treścią reguł gramatyki. Warto zwrócić uwagę na fakt, iż szczególnym przypadkiem rozpatrywanych gramatyk mogą być te operujące na strukturze drzewa (zamiast grafu). Znów umożliwia odwzorowanie relacji pomiędzy komponentami graficznymi modelowanego świata w formie struktury logicznej – tym razem z wykorzystaniem gałęzi drzewa. Umożliwia to wyrażene relacji fizycznego zawierania obiektów modelowanego świata. Gramatyki grafowe, podobonie jak miało to miejsce w przyapdku gramatyk kształtu, definiują instytucję komponentu terminalnego i nie terminalnego jako czynnika sterującego przebiegiem modyfikacji grafu. Tym razem jest to wierzchołek grafu. Etykietowanie wierzchołków grafu jako terminalnych. Wygenerowanie słowa gramatyki będzie polegało na otrzymaniu grafu zawierającego jedynie wierzchołki terminalne. *Język gramatyki grafowej* to tym samym zbiór wszystkich grafów zawierających wyłącznie wierzchołki terminalne, które można wyprowadzić z początkowago grafu (zwanego też czasami *aksjomatem*).

Istotnym dla rozważań w dalszych rozdziałach pracy zastosowaniem gramatyk grafowych było zaproponowane przez H.R. Nielsona i F. Nielsona użycie tzw. *drzew kształtów (shape-tree*)[Nie00]. Zaangażowano tu w procesie przetwarzania zbiory mnogościowe gramatyk grafowych i opracowano straregię szybkiego konstruowania gramatyk koniecznych do aplikowania przy rozwiązywaniu konkretnych problemów generacyjnych.

3.1.11 Algorytmy genetyczne zastosowane do kodowania kształtów

Innym podejściem dostarczającym rozwiązań problemu dopasowania LHS reguł kształtu do kształtów jest zastosowanie algorytmów genetycznych [Mit96]. Metoda bazuje na wprowadzeniu fazy wstępnej przetwarzania danego typu kształtów, polegającej na zdefiniowaniu tzw. *kodów kształtów* (*shape code*) [Kou00] – czyli genotypu, na którym algorytm genetyczny będzie operował. W praktyce kształt LHS staje się łańcuchem zawierającym kod reprezentujący kształcie. Metoda wymaga najczęściej kwantyzacji przestrzeni, w której zawarte są składowe kształtu (pojedyncze „kwanty" kształtu). Powoduje także konieczność zredukowania ilości możliwych typów tych składowych [Kou00]. Z drugiej strony wykorzystanie kodu kształtu umożliwia szybką walidację kształtów terminalnych (na przykład pod kątem „zgodności" z językiem gramatyki). Inną i znacznie ważniejszą zaletą jest możliwość fragmentacji kodu opisującego określony element kształtu – a w konsekwencji definiowanie wyspecjalizowanych reguł operujących na kształtach. Ułatwia to przygotowanie

zbioru reguł dla konkretnego zastosowania praktycznego. Cały czas jednak gramatyka kształtu operuje na ograniczonej liczbie kombinacji kształtów, jakie może przetwarzać.

Typowym przykładem zastosowania inżynieryjnego dla takiej modyfikacji gramatyki kształtu jest rozwijanie konstrukcji architektonicznych w „przestrzeni dyskretnej" (np. poprzez układanie w dyskretnej przestrzeni 3D sześcianów różnych typów). Tym samym możliwe jest wyprowadzanie różnego typu brył, reprezentujących np. układy apartamentów w bryle budynku czy gabinetów na piętrze biurowca. W tym przypadku pomieszczenia to zbiory przylegających do siebie sześcianów i opisanych kodem. Kod kształtu opisuje ułożenie sześcianów i stanowi materiał do przetwarzania algorytmem genetycznym. Może to być przykładowo ciąg wartosci binarnych odpowiadających kolejnym sąsiadującym sześciano. Gdy sześcian jest wypełniony (zawarty w kubaturze przyszłej bryły) – wartość logiczna bitu jest wsoka gdy nie – niska. Kolejne równolegle i ułożone obok siebie szeregi tych sześcianów reprezentowane są poprzez kolejne ciągi bitów. Te ciągi złaczane są w jeden kod według opracowanego indywidualnie klucza. Podobne kody zawarte są w LHS i RHS reguł kształtu.

3.1.12 Podejście ewolucyjne do rozbudowy gramatyk kształtu

Przez ostatnie kilkanaście lat w ogólnie panującej modzie na stosowanie „podejścia ewolucyjnego" jako metodę rozwiązania niemal każdego zagadnienia nie pominięto także gramatyk kształtu. Tutaj głównym celem do osiągnięcia jest ewolucyjnie sterowana modyfikacja puli reguł definiujących docelowo gramatykę. Testowanie nowo powstałej gramatyki zgodnie z podejściem ewolucyjnym polega na traktowaniu jej jako osobnika i porównywaniu generowanych a jej użyciem wyników. Z każdą iteracją algorytmu pula reguł ulegała modyfikacji (mutacja i krzyżowanie), możliwe więc było utworzenie nowego wariantu gramatyki kształtu (korekta w zbiorze kształtów nie terminalnych, a w konsekwencji powstanie różnic w gramatyce) [Ger07]. Nowa gramatyka była następnie testowana, zgodnie z wytycznymi podejścia ewolucyjnego.

3.2 Analiza zagadnień związanych z możliwościami rozwijania graficznych systemów generacyjnych.

Niniejszy podrozdział omówi konkluzje z przeglądu rozmaitych rozwiązań angażujących wiedzę w systemach generacyjnych. Na podstawie wiedzy o eksperymentach różnorodnych zespołów badawczych będzie można doprowadzić do ustabilizowania koncepcji przetwarzania łączącej zalety przetwarzania generacyjnego z projektowaniem hierarchicznym dla złożonych kształtów graficznych w euklidesowej przestrzeni trójwymiarowej. Rozważone zostaną teoretyczne podstawy systemu wykorzystującego takie założenia.

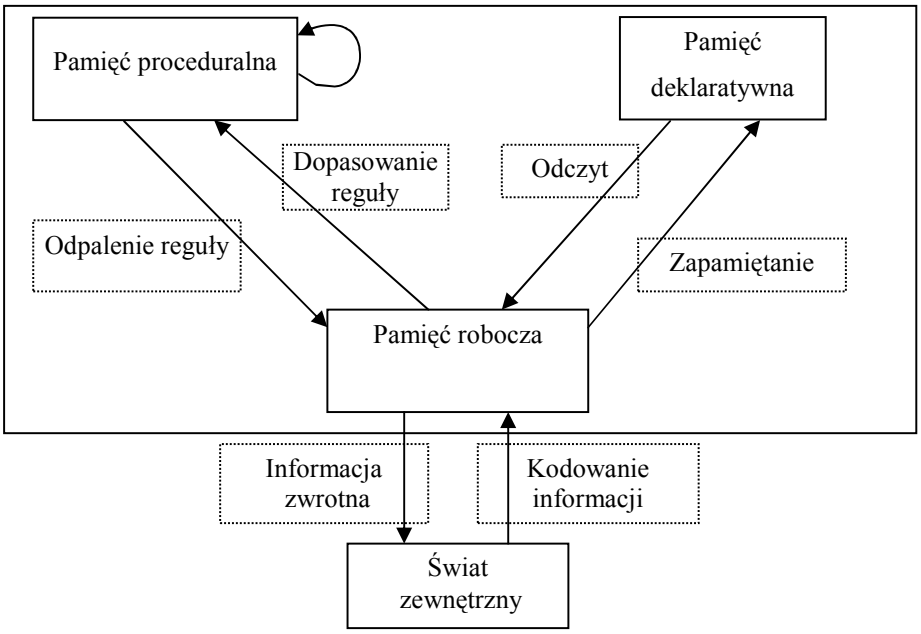
3.2.1 Reguły jako forma nośnika wiedzy

Ogólną definicję reguły kształtu zamieszczono w rozdziale drugim pracy. Wynika z niej pośrednio, iż reguła tworzy relację (powiązanie) między elementarnymi porcjami informacji, na których operuje system generacyjny. Z uwagi na formalnie zdefiniowaną strukturę, reguła jest łatwa do przechowywania i obróbki.

Klasyfikacja systemów regułowych (interpretowanych intuicyjnie jako systemy operujące na regułach) wyróżnia gałąź systemów rozwiązujących konkretne problemy oraz gałąź systemów ogólnych. Należy tu pamiętać o rozróżnieniu systemów regułowych od ekspertowych [Chd02]. Jest ono wyraźnie pomimo faktu, że spora część systemów regułowych to systemy ekspertowe. Istotnym wyznacznikiem systemu ekspertowego jest jego przeznaczenie, czyli analityczne rozwiązywanie problemów, dla których jest dedykowany. Treść systemu regułowego to jego struktura, prezentująca przykładowo sposób zakodowania reguł czy technikę ich przetwarzania [Mul96]. System regułowy definiuje więc ogólny model wnioskowania. W modelu tym dobrze jest rozwinąć symbolikę, na której reguły mogą bazować – co umożliwi dalszą redukcję zbędnej informacji [Chd02]. Symbol może być tu dowolnie reprezentowany, może też funkcjonować w hierarchii.

Stosowanie reguł daje także możliwość elastycznego zakodowania wiedzy postrzeganej. Z takimi procesami możemy mieć do czynienia, analizując duże ilości danych pod kątem pozyskania na ich podstawie informacji okrojonej, lecz uogólnionej. Posiadając zdefiniowaną strukturę bazy wiedzy, potrzebny jest jeszcze jedynie model procesu poznawczego, według którego możliwe będzie zbudowanie wypełnionej treścią bazy wiedzy. Proces taki może funkcjonować w pełni automatycznie, sukcesywnie rozbudowując zasoby wiedzy. Dał temu

dowód choćby J. Anderson, proponując model poznawczy [And83] o szczegółowo ujętej architekturze, znany pod nazwą ACT*. Wypracował on praktyczne rozwinięcie własnej teorii (ACT) dotyczącej technik poznawczych stosowanych przez człowieka (postrzegania świata oraz wyrażania spostrzeżeń) [And76]. Jego wytyczne zostały wzięte pod uwagę w kolejnych pracach, także pracach autora [Tur05]. Będą też przydane do rozważań na temat pozyskiwania danych nadających się do obróbki wielowarstwowymi gramatykami kształtu w niniejszym opracowaniu. Podstawowym założeniem tego akurat modelu jest zastosowanie pamięci krótkotrwałej (jako "notatnika") i dwóch rodzajów pamięci długotrwałej: pamięci deklaratywnej, w postaci sieci semantycznej, oraz pamięci proceduralnej, jako zbioru reguł. Każdy element pamięci długotrwałej (zarówno węzeł sieci jak i reguła) posiadają atrybut aktywności. Reguły mogą powodować uaktywnianie się kolejnych komórek pamięci, adresowanych symbolami z konkluzji tych reguł. W węzłach sieci kodowane są konkretne dane elementarne podlegające obróbce, takie jak łańcuchy znaków, liczby itp. Pamięć krótkotrwała zawiera informację uzyskaną dzięki percepcji (pobieraną ze środowiska), wskazania na elementy aktywne (w konsekwencji odpalania reguł) oraz wskazania na elementy aktywne wcześniej. Ponadto aktywacja może rozchodzić się po sieci semantycznej – jako funkcja wagi przypisanej elementom. Waga ta wzrasta wraz z użytkowaniem elementu (gdy przykładowo był konkluzją użytej reguły), waga maleje – z brakiem użytkowania. Dodatkowo – wagi brane są pod uwagę w procesie odpalania reguł – wygrywa reguła o największej wartości wag jej przesłanek. Dopasowując reguły z pamięci proceduralnej do aktywnych symboli system uzyskuje zbiór reguł, które można odpalić. Po odpaleniu wybranej reguły aktywuje symbole będące jej konkluzją. Następnie ustala nowe aktywne elementy sieci semantycznej – śledząc rozchodzenie się aktywacji w sieci od uaktywnionych przez odpalenie reguł symboli. Elementy te wpisuje zaznacza pamięci krótkotrwałej. Generalnie zbiór reguł jest kształtowany niezależnie od danych pochodzących z percepcji, a system jedynie przetwarza te dane na podstawie reguł. Została jednak dopuszczona możliwość "uczenia się" nowych reguł na podstawie wielokrotnych wystąpień analogicznego przebiegu wnioskowania. Dodatkowo w bazie wiedzy umieszczane są reguły, prowadzące do celu wnioskowania – dzięki którym system może oddziaływać na świat zewnętrzny produkując w konsekwencji procesu wnioskowania informację zwrotną (Rysunek 3.1).



Rysunek 3.1: Schemat architektury „ACT*" [And83].

Jak widzać przetwarzanie regułami może być stosowane także w bardzo zróżnicowanych strukturalnie architekturach hybrydowych. Przedmiotem przetwarzania nadal jest wiedza określająca model wybranej dziedziny świata rzeczywistego (zewnętrznego).

3.2.2 Problem przetwarzania nie sklasyfikowanych kształtów graficznych

Głównym problemem przetwarzania sygnałów pochodzących z naturalnego środowiska jest brak możliwości deterministycznego i formalnego klasyfikowania tych sygnałów. Czynników będących źródłem takiego problemu jest bardzo wiele – począwszy od zaszumienia pozyskiwanego materiału, przez mnogość jego form, na problemach skalowalności metody przetwarzającej kończąc. Konieczne są zatem rozwiązania umożliwiające usprawnienie procesu rozpoznawania kształtu dostarczanego do gramatyki. Jedno z pierwszych rozwiązań dostarczył Krishnamurti [Kri92] – rozwijając mechanizmy gramatyki kształtu o funkcjonalność w zakresie ich rozpoznawania. Rozpoznawanie ograniczało się do identyfikacji linii prostych, a na późniejszych etapach prac - fragmentów łuku oraz *kształtów graficznych* typu *spline*. Obliczenia na *kształtach graficznych* były realizowane na podstawie wspólnych „koordynat kształtów", przekazywanych także do *reguł kształtu*. Spowodowało to konieczność zmodyfikowania definicji produkcji *gramatyki kształtu*. Krishnamurti wprowadził tak zwane *parametryczne gramatyki kształtu* (*parametric shape grammars*), w których każda reguła typu

A → B może być użyta do zdefiniowania reguł wyprowadzonych w postaci g(A) → g(B) (gdzie g jest tzw. *funkcją parametryzacji*). W takiej sytuacji klasyczne „odpalenie" reguły A → B dokonujące transformacji C′ = [C − t(A)] + t(B) zostanie zamienione na C′ = [C − t(g(A))] + t(g(B)). Funkcja g odpowiada za konwersję kształtu – odpowiednika do postaci dopasowywalnej do LHS reguły kształtu. Dzięki temu wiele kształtów odległych od siebie (w rozumieniu ich reprezentacji graficznych), ale jednak unifikowanych przez g() będzie mogła zostać przetworzone przez tą samą regułę. Funkcja g() została w metodzie Krishnamurti ujęta dość abstrakcyjnie. Nie zaproponowano żadnej metodyki usprawniającej projektowanie takiej funkcji dla określonych typów kształtów. Wyprowadzono jednak ogólną koncepcję oddzielenia abstrakcyjnie ujętych informacji o kształcie (przetwarzanych w tym przypadku przez regułę kształtu) od informacji o jego usytuowaniu (podawanej jako „koordynata kształtu" do funkcji g()), co jest istotnym sygnałem dla kierunku rozważań zawartych w następnych rozdziałach pracy. W przypadku opracowania Krishnamurti parametryzacja funkcji g() wykonywana jest dopiero na poziomie implementacji metody, a dane do niej pochodzą od użytkownika (jego interwencja jest wymagana na każdym kroku przetwarzania). Kontynuacja rozważań na temat klasyfikacji kształtów została podjęta w opracowaniu McComarcka [Mcm02], idąc w kierunku dodefiniowania pod-kształtów interpretowanych dodatkowo przez funkcję g() .

Patrząc na kształty graficzne w algebrze U_3 każdy z nich będzie skończonym (lecz także ewentualnie pustym) zbiorem tzw. *elementów podstawowych* (*basic elements*) – linii, punktów, powierzchni płaskich itp. Definiuje się dodatkowo tzw. *maksymalny element podstawowy* (*maximal basic element*), jako kształt który nie może być zbudowany z innych elementów podstawowych. Jest to istotne z punktu widzenia algorytmu rozpoznawania kształtów, gwarantując ich deterministyczną identyfikację. Każdy kształt graficzny A jest zatem zbiorem nieposortowanych *elementów podstawowych*: $A = \{a_1, a_2, \ldots, a_n\}$. Operatory sumy i różnicy kształtów pozostają niezmienne [Sti80]. Wprowadzenie elementów podstawowych oraz zbiorów definiujących kształty znacznie usprawnia identyfikację kształtów w dowolnym materiale wejściowym. Problem identyfikacji sprowadza się wówczas do poszukiwania (stosunkowo prostymi transformacjami) elementów podstawowych w tym materiale i dopasowywania ilości ich wystąpień do zawartości zbiorów definiujących kształty graficzne. Z kolei połączenie reprezentacji opisanej wyżej reprezentacji kształtów z zastosowaniem systemu koordynat elementów kształtu umożliwi kolejne uściślenie definicji poszukiwanego w materiale kształtu. Powstała konstrukcja zawierać będzie zbiór elementów podstawowych oraz dodatkową informację o wzajemnym ich usytuowaniu.

3.2.3 Wykorzystanie wiedzy rozproszonej w wielu zbiorach produkcji

Chaotycznie upakowany zbiór reguł kształtu nie dostarczy informacji koniecznej do przeprowadzenia procesu generowania jakiegokolwiek złożonego obiektu. Jak wspomniano we wcześniejszych rozdziałach, gramatyka definiująca jedynie transformacje tego typu kształtów (pod postacią reguł) umożliwi ich użycie bez zachowania determinizmu. Możliwe jest więc użycie wielu reguł w jednym momencie – tych, które definiują dostatecznie liberalne uwarunkowania stosowalności reguły (po stronie LHS). Z kolei w przypadku zaostrzenia warunków (celem minimalizacji ilości takich możliwości) szybko dojdzie do sytuacji, kiedy żadna z reguł nie będzie mogła zostać użyta. To spowoduje zatrzymanie przetwarzania danej gramatyki.

Głównym założeniem, mającym zlikwidować opisany problem, jest wprowadzenie dodatkowej klasyfikacji reguł, która będzie podstawą do kontroli ich dopasowania w procesie przetwarzania gramatyki kształtu. Ideą wzorcową jest tu analogia do relacji definiowanych w metodykach podejścia obiektowego pomiędzy klasami obiektów [Kni99]. Osobną kwestią będzie naturalnie problem pozyskiwania samych produkcji (reguł). Najprostszą z metod jest budowa bazy wiedzy o regułach na podstawie proceduralnie ujętej sekwencji operacji, koniecznych do przeprowadzenia na przetwarzanym materiale podczas modyfikacji gramatyką. Dodatkowa klasyfikacja reguł wymusza wprowadzenie nowej techniki przetwarzania gramatyką, co będzie jednym z przedmiotów rozważań w daszej części pracy.

3.2.4 Porządkowanie hierarchii obiektów i ich odpowiedniki znaczeniowe – podejście ontologiczne

Słownik symboli identyfikujących obiekty graficzne (licznie występujące w każdej z przetwarzanych dziedzin) jest z oczywistych względów obszerny i nieuporządkowany. Budzi to pewien problem, gdyż konieczne są środki umożliwiające wyrażenie znaczenia danego obiektu w jego dziedzinie, powiązania go z identyfikatorem postrzeganym dla eksperta w dziedzinie (nazwą) oraz zakodowania ewentualnych relacji łączących dany obiekt z innymi. Rozwiązanie problemu może przynieść teoria tzw. ontologii [Smi04], zakładająca wprowadzenie regulacji w przestrzeniach nazw obiektów precyzyjnie odpowiadających postawionym wyżej zapotrzebowaniom. Istnieje już kilka opracowań wprowadzających zastosowania ontologii w grafice. Większość z nich zakłada wykorzystanie porządku

gwarantowanego przez ontologie dla klasyfikowania przetwarzanych obiektów. Celowym więc wydaje się przeprowadzenie rozważań nad możliwością analogicznego zastosowania ontologii w rozwiązaniu bieżącego problemu.

Generalnie *ontologią* (w naukach informatycznych) nazywamy formalną reprezentację wiedzy jako zbiór terminów definiowanych w ramach domeny, oraz relacji pomiędzy tymi terminami [Gru93]. Etymologia samego terminu nadal ma źródła w filozoficznym pojęciu „sposobu istnienia" lub „sposobu przedstawiania" danego bytu, zwracając także uwagę na związek pomiędzy ontologią a logiką. Kategoriom semantycznym zmiennych przyporządkowywane są bardzo podobne kategorie ontologiczne w tworzonym przez ontologię modelu semantycznym (znaczeniowym). W ramach kategorii ontologicznej możliwe jest klasyfikowanie pojęć zbliżonych, lub także identycznych – lecz identyfikowanych różnymi słowami języka naturalnego [Per07]. Wymiana wiedzy stanowiącej materiał do identyfikacji obiektów graficznych (a w konsekwencji reprezentowanych prze nie obiektów fizycznych) powinna być środkiem bardzo przydatnym. Może być użyta przy prowadzeniu procesów konsolidacji różnych gramatyk przetwarzanych w ramach graficznych systemów generacyjnych.

4 Cele i tezy pracy

Omówione we wcześniejszych rozdziałach motywacje oraz studia prowadzone w dziedzinie obecnie istniejących praktycznych zastosowań graficznych systemów generacyjnych pozwalają na ramowe sformułowanie celu pracy o następującej treści:

" Wyspecyfikowanie metodologii, która umożliwi deterministyczne, automatyczne i niezależne od dziedziny obiektów 3D projektowanie systemów przetwarzających złożone kształty 3D w oparciu o teorię gramatyk kształtu i przetwarzanie hierarchiczne. W rozwinięciu tej metodologii - opracowanie szeregu rozwiązań, które ułatwią wykorzystanie projektowanych transformacji w zastosowaniach inżynieryjnych."

Cel będzie osiągany poprzez opracowanie metodologii wytwarzania systemów generacyjnych zawierających także konstrukcje hierarchiczne. W szczególności powstaną opracowania oraz prowadzone będą badania w następujących kierunkach:

- opracowanie podstawowych założeń metodologii rozwijania systemów przetwarzających obiekty 3D, występującej w pracy pod nazwą Kaskadowych gramatyk kształtu (K-GK) i opartej na klasycznej teorii *gramatyk kształtu* oraz projektowaniu hierarchicznym,
- opracowanie tzw. *modelu klas kształtów* oraz modelu hierarchicznego dla owej metodologii,
- opracowanie definicji gramatyki K-GK wraz z systemem produkcji przetwarzających hierarchicznie uporządkowaną wiedzę o kształtach graficznych,
- zdefiniowanie metodologii tworzenia wielowarstwowych, komponentowych systemów przetwarzania kształtów graficznych wykorzystujących w transformacjach K-GK,
- stworzenie bazującego na XML języka umożliwiającego reprezentowanie baz wiedzy K-GK,
- stworzenie systemu wysokowydajnej graficznej i trójwymiarowej prezentacji tworzonych kształtów graficznych - zintegrowanego z elementami modelu K-GK,

- określenie ontologicznej reprezentacji wiedzy przetwarzaniej przez K-GK – umożliwiającej rozbudowę powiązań pomiędzy wiedzą o różnych dziedzinach kształtów.

Pomyślne przeprowadzenie badań potwierdzi zasadność tezy głównej pracy, określonej następująco:

"Możliwe jest dokonanie takiej przebudowy modelu klasycznych gramatyk kształtu, aby cały proces decyzyjny dotyczący stosowania ich reguł przebiegał w pełni deterministycznie, z uwzględnieniem rozbudowanej wiedzy opisującej daną dziedzinę projektową, oraz był równocześnie prowadzony bez konieczności ingerencji jakiegokolwiek czynnika ludzkiego."

W trakcie rozważań możliwe będzie wsparcie tez pobocznych:

„Zastosowanie proponowanego modelu Kaskadowych gramatyk kształtu (K-GK) w połączeniu z założeniami teorii ontologii dziedzinowych umożliwi swobodne pobieranie wiedzy o kształtach graficznych od ekspertów w danej dziedzinie, oraz kolektywne wykorzystanie tej wiedzy w wielu innych dziedzinach projektowania dla potrzeb grafiki."

„Wiedza o obiektach graficznych, gromadzona w modelu Kaskadowych gramatyk kształtu (K-GK), może zostać w całości wyrażona przy użyciu języków opartych na znacznikach - co ułatwi jej wykorzystanie w systemach przetwarzających obiekty graficzne."

5 Metodyka rozwiązania

W poprzednich rozdziałach przedstawiono szereg zagadnień teoretycznych nawiązujących do problematyki rozwijania grafiki komputerowej z użyciem metod formalnych i pół-formalych. Ponadto zaprezentowano ciekawsze próby wykorzystania aparatu formalnego do wspierania procesu przetwarzania kształtów w ich konkretnych dziedzinach. Większość z nich podąża ścieżką, zakładającą jedynie wykorzystanie konkretnej klasycznej gramatyki kształtu Stiny'ego do rozwiązania pojedynczego i konkretnego problemu w danej dziedzinie życia.

Metodyka zaproponowana w pracy, której rozwinięciu zostanie poświęcony bieżący rozdział, nie zakłada ograniczenia rozważań do konkretnej gramatyki kształtu czy też konkretnej dziedziny kształtów graficznych. Przykłady z poprzedniego rozdziału dobitnie pokazały, iż proponowanie kolejnej gramatyki kształtu, która funkcjonując w bardzo wąskiej dziedzinie dostarczy jedynie w zastosowaniach praktycznych lepsze lub gorsze wyniki mija się z celem. Z kolei proponowanie gramatyki kształtu zawierającej reguły uogólnione całkiem uniemożliwi wykorzystanie jej w zastosowaniach inżynieryjnych. Dlatego głównym celem będzie opracowanie i zaproponowanie kompleksowego systemu do zarządzania gramatykami kształtu, operującymi jednocześnie w różnorodnych dziedzinach (tym samym - nad najróżniejszymi alfabetami kształtów). Zaprojektowany zostanie model klas kształtów, na którym możliwe będzie umieszczanie przestrzeni symbolicznych identyfikującej poszczególne grupy kształtów. Powstanie także model samej rozszerzonej gramatyki, będącej narzędziem dla rozwiązań wspomagających generacyjne przetwarzanie kształtów graficznych. W konsekwencji – powstanie metodologia tworzenia autonomicznych systemów przetwarzania. Bazując na wzajemnych powiązaniach angażujących kształty graficzne, gramatyki kształtu oraz zbiory graficznych transformacji kształtów metodologia ta będzie postulowała utworzenie nowego systemu przetwarzania, operującego na modelu *K-GK*.

5.1 Przestrzeń funkcjonowania K-GK

Jak wspomniano w poprzednim rozdziale - obiekty graficzne, które potencjalnie można przetwarzać, w znacznej większości przypadków są bytami reprezentującymi faktyczne obiekty fizyczne świata rzeczywistego. Modelowanie polegać tu będzie głównie na umieszczaniu takich obiektów w n-wymiarowej przestrzeni wirtualnej. Powstałe kompozycje wiążą ze sobą obiekty, które w świecie rzeczywistym także są ze sobą powiązane. Powiązania mogą wytwarzać uporządkowane listy elementów, na których zawrzemy obiekty postrzegane przez osobę modelującą (eksperta). Takie listy koncentrują zatem obiekty, które będą posiadały zbliżoną strukturę, będą komponentami innego (wspólnego) obiektu itp. Mogą to być przykładowo wszystkie obiekty wchodzące w skład budynku, pojazdu, czy wnętrza – czyli należące do określonej opisowo zdefiniowanej dziedziny. Zbiór takich obiektów (ze względu na występowanie wspomnianych powiązań) można zatem nazwać *dziedziną obiektów*, zaś zbiór kształtów reprezentujących takie obiekty w przetwarzanym modelu – *diagramem*. Podjęcie próby formalnego zdefiniowania dziedziny kształtów mija się z celem, gdyż (jak wspomniano w poprzednim rozdziale) różnorodność kształtów i powiązań możliwych do zdefiniowania przy projektowaniu konkretnych zastosowań inżynieryjnych jest ogromna. Dopiero wiedza zdefiniowana przez eksperta w danej dziedzinie obiektów będzie mogła zostać ustrukturyzowana. Ważne jest jednak naturalnie to, aby każdą dziedzinę obiektów można było opisać wiedzą.

Postawione tezami pracy wymagania funkcjonalne dla rozwiązania wspomagającego przetwarzanie z użyciem gramatyk kształtów zakładają, iż projektowane rozwiązanie będzie:

- przeznaczone do modyfikacji komponentów graficznych umieszczonych w dwuwymiarowej lub trójwymiarowej przestrzeni Euklidesa,
- uniwersalne wobec dziedzin przekształcanych obiektów,
- operowało na modelu abstrakcyjnym i rzeczywistym (dziedzinowo zależnym) z rozłączeniem tych procesów,
- rozszerzalne o nowe operacje modyfikujące obiekty,
- wykorzystywało informację opisującą dziedziny obiektów graficznych,
- uniezależnione od sposobu reprezentacji obiektów graficznych w przestrzeni Euklidesa.

Przy takich założeniach można skupić się na projektowaniu systemu, który spełniał będzie stawiane wymagania (funkcjonalne i poza funkcjonalne).

Transformacje z wykorzystaniem modelu klas będą umożliwiały sekwencyjne angażowane wielu gramatyk w procesach wspomagających obróbkę grafiki trójwymiarowej, wykorzystując kolektywnie wiedzę szczątkową pochodzącą z różnych źródeł. Zostanie zdefiniowana specjalna arytmetyka, stanowiąca pomost pomiędzy klasami kształtów i relacjami wykorzystywanymi dotychczas przez „klasyczne" gramatyki kształtu w procesach dopasowania reguł (usytuowanie kształtów, wymiary etc).

Problematyka identyfikacji kształtu wyjdzie daleko poza rozważanie treści zbioru kształtów odpowiadających słowom języka pojedynczej gramatyki.

5.2 Definicje pojęć pomocniczych

Pojęcia podstawowe, jak choćby ogólny termin *kształtu graficznego (geometrycznego),* zostały już sprecyzowane w pierwszym rozdziale pracy. Do rozbudowy koncepcji *K-GK* konieczne jednak będzie zdefiniowanie kolejnych, mających zastosowanie już wyłącznie w odniesieniu do bieżącego opracowywanych rozwiązań. Podobnie jak to zostało ustalone w rozdziale drugim, każde odwołanie do zdefiniowanego terminu występujące w późniejszych rozdziałach pracy będzie oznaczone tekstem *pochylonym*.

W pierwszej kolejności samo pojęcie *kształtu* musi zostać rozszerzone. Kształt należy osadzić w przestrzeni Euklidesa o wymiarze zredukowanym. Ponadto konieczne będzie zaprojektowanie modelu klas, przy pomocy którego kształty będą klasyfikowane, oraz rozszerzenie definicji kształtu o nowe atrybuty.

Definicja 5.1:

Kształt k w rozumieniu teorii K-GK będzie czwórką:

$$k = < \lambda, P, \gamma, K>$$

gdzie

λ jest *klasą kształtu*, należącą do zbioru *klas kształtów* C,

γ jest dwuwartościowym stanem kształtu (określającym czy jest on *terminalny*),

P jest m-elementowym zbiorem meta-danych. Zawartość zbioru P określa usytuowanie kształtu w przestrzeni Euklidesa,

K jest zbiorem dowolnych identyfikowalnych obiektów, posiadających atrybuty, usytuowanych w *n*-wymiarowej przestrzeni Euklidesa. ■

Od tego momentu każdorazowe wystąpienie w tekście terminu „*kształt*" odnosić się będzie go *kształtu* w rozumieniu teorii *K-GK* , nie zaś *kształtu* definiowanego w rozdziale 2.2.

Kształt będzie jednoznacznie przypisywany do klasy kształtu.

Definicja 5.2:

Klasą kształtu będziemy nazywali jednoznacznie określony zbiór cech danego kształtu.

Definicja 5.3:

Kształt posiadający reprezentację graficzną (z racji przypisania do rozpoznawalnej *klasy kształtu,* która nie jest *abstrakcyjna*) będziemy nazywali *kształtem graficznym.*

Zmieniamy zatem interpretację terminu z rozdziału 2.2 na analogiczną, lecz w rozumieniu teorii *K-GK* . Definicja 5.6 *klasy abstrakcyjnej* zostanie zaprezentowana w rozdziale 5.4.

Indywidualne traktowanie wyglądu *kształtu graficznego* umożliwi pełne uniezależnienie od niej wszelkich projektowanych rozwiązań. Na niższym poziomie abstrakcji interpretowane jest tzw. usytuowanie obiektu w przestrzeni Euklidesa. Meta-dane ze zbioru P określające owo usytuowanie mogą mieć rozmaitą postać. P może być przykładowym zbiorem n-tek, gdzie n będzie określało wymiar przestrzeni Euklidesa. Wówczas poszczególne n-tki będą definiowały kolejnowektory pozycji, rotacji, skalowania obiektu w przestrzeni trójwymiarowej. Równie dobrze P może być jednoelementowy i zawierać macierz transformacji (o wymiarach 4x4) dla obiektów przemieszczanych w przestrzeni Euklidesa. Postać elementów zbioru P nie będzie uzależniać przebiegu procesu dopasowania reguł.

Od tego momentu, termin *kształt* będzie oznaczał zaprezentowany właśnie jego wariant rozszerzony.

Definicja 5.4:

Kształt jest *nie terminalny*, jeśli element γ tego kształtu jest fałszywy. W przeciwnym wypadku *kształt* jest *kształtem terminalnym*. ■

Pojęcia te będą miały związek z wprowadzonymi w następnych, a w konsekwencji z możliwością przetwarzania kształtu z użyciem wspomnianych reguł.

Definicja 5.5:

Diagramem kształtów (lub *modelem kształtów*) będzie nazywana uporządkowana lista kształtów posiadających swoje klasy kształtów oraz poiadających z meta-dane tych kształtów przetwarzanymi w konsekwencji prowadzenia transformacji *diagramu kształtów*. ■

Przed rozpoczęciem przetwarzania diagramu kształtów obiekt ten będzie zawierał jedynie tzw. *kształt początkowy (lub kształty początkowe),* wywodzący się pojęciowo z teorii klasycznych gramatyk kształtu. *Kształt* ten jest nie terminalny.

5.3 Ogólna koncepcja rozwiązania

Generalną myślą przewodnią warunkującą zastosowanie dużo bardziej skomplikowanego i wieloetapowego procesu przetwarzania *kształtów* było odseparowanie uwarunkowanej dziedzinowo transformacji sygnałów (*kształtów graficznych*) od modelu zadanego formalnie. Przetwarzanie obiektów w danej dziedzinie będzie zatem polegać na stosowaniu transformacji dedykowanych dla tej dziedziny. Proces decyzyjny dotyczący wykorzystania określonych transformacji będzie dyktowany treścią formalnie zdefiniowanej informacji. Przy tak określonym podejściu będzie możliwe definiowanie gramatyk uniwersalnych, mogących w konsekwencji operować na różnych dziedzinach *kształtów* – co w przypadku klasycznych gramatyk kształtu było rzeczą nieosiągalną (brak istnienia dopuszczalnych transformacji identyfikujacych pod-kształt).

Przestrzeń projekcji komponentów graficznych będących przedmiotem przetwarzania to głównie *przestrzeń 3D* lub *przestrzeń 2D*. Transformacje powinny być elementami zbioru mnogościowego, którego treść będzie każdorazowo uzależniona od dziedziny przetwarzanego materiału. Możliwe powinno być tworzenie powiązań pomiędzy różnymi dziedzinami naturalnej wiedzy.

Formalizm służący do wspomagania przetwarzania będzie wymuszał nie tylko modyfikacje reprezentacji graficznej, ale także korekty symbolicznego znaczenia obiektów przetwarzanych - modyfikując wzajemne relacje między nimi. Wyjdzie zatem poza określanie jedynie technicznych aspektów transformacji *kształtów*. Finalnie wprowadzi do procesu przetwarzania trzy poziomy abstrakcji:

- najwyższy - bazujący na przygotowanej przez eksperta dziedzinowego abstrakcyjnej wiedzy o typach obiektów przetwarzanych oraz powiązaniach logicznych między tymi typami,
- pośredni - rozważający reguły oddziaływania obiektów danego typu na inne – z uwzględnieniem abstrakcyjnie ujętych transformacji tych obiektów w jeszcze nieokreślonej przestrzeni projekcji będącej ich środowiskiem,
- najniższy (rzeczywisty) - który będzie operował na konkretnych reprezentacjach graficznych przetwarzanych *kształtów* i w ustalonej już przestrzeni - przetwarzając je przy użyciu klasycznych *gramatyk kształtu.*

Przy tak zdefiniowanych założeniach wstępnych procedura analizująca informacje o logicznym *diagramie kształtów* i o ich reprezentacjach graficznych powinna przetwarzać *kształty* z użyciem mechanizmów formalnych. Nie narzuci jednak ograniczeń związanych ze specyfiką reprezentacji graficznej obiektów, których odpowiedniki logiczne są przetwarzane.

5.3.1 Bazy wiedzy o regułach kształtu

Metodologia K-GK zakłada stworzenie nowej metodyki przetwarzania z użyciem gramatyki kształtu, bazującej na istnieniu wielu rodzajów powiązań pomiędzy elementami zbiorów tej gramatyki. Powiązania te dają możliwość poszukiwania w bazie wiedzy możliwości realizowania rozbudowanych transformacji w sposób dużo bardziej złożony i dopasowany. Wartościowy proces dopasowywania i uruchamiania reguł powinien według autora charakteryzować się następującymi właściwościami:

- powinien móc przebiegać wielokrotnie dla każdej z reguł (reużywalność reguł),
- musi bazować na wykorzystaniu różnych wariantów reguł modyfikujących *kształty*, dając możliwość rozszerzenia ich definicji o nowe atrybuty (kontrola rodzaju prowadzonych operacji elementarnych),

- powinien operować na specjalnie zdefiniowanych modelach abstrakcyjnych charakteryzujących dziedzinę przetwarzanego *diagramu kształtów* (możliwość prowadzenia kategoryzacji jakościowej elementów *diagramu kształtów*),
- *diagram kształtów* powinien przetwarzać transformacjami interpretującymi także kwalifikację logiczną *kształtów*, a niekoniecznie jedynie ich reprezentację graficzną (daleko idące ułatwienie przetwarzania),
- powinien wykorzystywać reguły, których konstruowanie jest możliwe w oparciu o wiedzę eksperta pochodzącą z dziedziny obiektów przetwarzanych (ułatwienie definiowania wiedzy),
- powinien przetwarzać także *kształty* nie posiadające reprezentacji graficznej – ale będące nośnikiem informacji o logicznych powiązaniach innych *kształtów* (brak konieczności renderowania obiektów niematerialnych),
- powinien zakończyć przetwarzanie zwracając deterministyczny wynik, który posiada możliwą do wyrenderowania reprezentację graficzną (przewidywalność).

Konieczne zatem będzie zdefiniowanie złożonego modelu opisującego strukturę informacji o *kształtach* i możliwych do prowadzenia transformacjach, określając je formalnie. Postępowanie przy pracach projektowych można dostosować do wytycznych zadanych w teorii tzw. wnioskowania diagramowego [Kul03], gdzie we wspomnianym procesie wnioskowania można wyróżnić etapy kodowania, przetwarzania, dekodowania oraz dywagować nad ich granicami. Skład danych zakodowanych będzie bazą wiedzy, eksponując cechy dla baz wiedzy charakterystyczne [Pan93]:

- możliwość przetwarzania encji, które mogą być dowolnie opisane i rozbudowywane,
- elastyczne kodowanie wszelkiej informacji,
- automatyczne generowanie treści faktów – z użyciem transformacji, które wygenerują je na podstawie analizowanego obiektu wejściowego,
- dowolne rozszerzanie formatu danych przechowywanych przez system – już po jego uruchomieniu - odbywające się drogą tworzenia nowych encji o nowej treści.

5.4 Model K-GK

Jak wiadomo, każdy *kształt* mogący posiadać swój ewentualny odpowiednik w LHS *reguły kształtu* powinien być jednoznacznie rozróżnialny w procesie dopasowania. W proponowanych rozwiązaniach za pośrednictwem dodatkowej informacji opisującej będzie zatem utożsamiany z tzw. *klasą kształtu,* charakteryzującą jego właściwości. *Kształt* może jednak ewoluować – między innymi zmieniając swoją klasę. *Kształt* może zmienić klasę w przypadku istnienia relacji pomiędzy jego aktualną klasą, a ewentualną nową klasą dla tego *kształtu*. Możliwość rozróżnienia *kształtów* identyfikowalnych poprzez klasy umożliwi prowadzenie ich ciągłych przekształceń, zarówno w kierunku „od ogółu do szczegółu" (co odpowiadało by zmianie klasy *kształtu* na klasę wyprowadzoną) jak i w kierunku odwrotnym. Drugi wariant pozwoli na rozszerzenie zakresu poszukiwań metod przetwarzania *kształtu* – w przypadku braków w lokalnej wiedzy o jego naturze. W konsekwencji ewidentnie widocznej gradacji poziomu szczegółowości obrabianych obiektów będzie można wprowadzić grupowanie samych reguł *kształtu*. W swoich LHS będą one posiadały *kształty* przyporządkowane do określonych „grup szczegółowości". Reguły *kształtu* będą zatem operować na materiale o różnym stopniu uszczegółowienia – od materiału całkowicie abstrakcyjnego (swoją drogą deklarowanego za pomocą klas abstrakcyjnych) do uszczegółowionego – reprezentującego detale *diagramu kształtów*.

Na pierwszy rzut oka klasyfikacja składników LHS zdefiniowana powyżej spowoduje podział bazy wiedzy na *n* list zawierających reguły. Dodatkowo każdy z podzbiorów stworzyłby zamkniętą bazę wiedzy produkującą *kształt* wyjściowy na podstawie wejściowego. *Kształt* wyjściowy byłby następnie przekazywany do obróbki z zastosowaniem kolejnej listy reguł. Tym samym cały proces przetwarzania reguł zostałby podzielony na (niezależne) etapy, realizowane kolejno. Tak jednak się nie stanie – ponieważ LHS każdej z reguł może być listą wielu elementów, w tym także elementów należących do różnych „grup szczegółowości" obiektu (gramatyka będzie kontekstowa). W konsekwencji proponowana technika nie spowoduje rozwarstwienia bazy wiedzy, co stanowi jej istotną zaletę. Umożliwi za to kontrolę procesu uruchamiania reguł sprawiając, że przetwarzany z użyciem takiej bazy wiedzy obiekt nadal będzie mógł ewoluować.

Posiadanie w bazie wiedzy takiego uporządkowania pozwoli przykładowo na:

- scalenie zgubionego wcześniej elementu graficznego z już istniejącym produktem (będącym *kształtem* dopracowanym, czyli znacznie już przetworzonym),
- cofnięcie procesu kompozycji *kształtów* po napotkaniu kolidującego i wcześniej pominiętego *kształtu*,
- rozbicie produkowanego obiektu (z podobnych przyczyn) itp.

Takie właśnie operacje elementarne będą zawsze konieczne do prowadzenia przy swobodnym projektowaniu algorytmów przetwarzających dużą ilość znacznie zróżnicowanych jakościowo obiektów.

5.4.1 Model klas w K-GK

W podejściu obiektowym do programowania, stosowanym do projektowania jakichkolwiek systemów informatycznych często stosuje się definicję klasy. Klasa może być łączona ogólnie znanymi relacjami z innymi klasami - umożliwiając wprowadzenie hierarchii klas. Na podstawie klas tworzone są ich instancje, będące nośnikami danych zawartych w systemie oraz (poprzez funkcjonalność swoich metod) określające czynności możliwe do wykonania.

Reguła „klasycznej" gramatyki kształtu definiowana jako LHS→RHS określa możliwą do wykonania operację na materiale graficznym. Operacja wiązać się będzie z przetworzeniem *kształtu* określanego w LHS na *kształt* zdefiniowany w RHS. Operacji takich może być wiele. Mamy zatem dane i zbiór możliwych operacji na nich poączone w jedną jednostkę logiczną. Daltago ją właśnie, wobec widocznej już analogii do modelu klas podejścia obiektowego przy programowaniu, będziemy identyfikowali pod nazwą *klasy kształtu*. W tym momencie można także rozpocząć rozważania na temat przydatności relacji pomiędzy *kształtami*, analogicznych do tych, które występują pomiędzy klasami w podejściu obiektowym. Na pewno wprowadzenie *klasy kształtu*, dostarczy dodatkowej informacji dla procesu przetwarzającego *gramatykę kształtu*. Istnienie takiej informacji umożliwi wzbogacenie procesu decyzyjnego, który orzeka o uruchomieniu reguły w procesie generowania *kształtu graficznego*.

5.4.2 Gramatyki kształtu w klasach kształtów

Gramatyka kształtu (zgodna z Definicją 2.5) nie będzie encją traktowaną indywidualnie, lecz komponentem *klasy kształtu*. Patrząc na *klasę kształtu* poprzez pryzmat modelu klas języka programowania – gramatyka kształtu będzie odpowiadała konstruktorowi klasy.

Gramatyka ta reprezentuje wiedzę precyzującą czynności mające na celu wygenerowanie finalnej reprezentacji graficznej *kształtu*, który reprezentuje dana *klasa kształtu*.

Tym samym wiedza zawarta w *gramatyce kształtu* będzie kolektywna – wiele dodawanych do niej gramatyk będzie wykorzystywane do wygenerowania końcowej reprezentacji graficznej *diagramu kształtów*.

Podstawowym celem i zaletą umieszczenia *gramatyki kształtu* w każdej z *klas kształtu* oraz wprowadzenia mechanizmu ich kaskadowego uruchamiania jest utrzymanie zarówno możliwości przetwarzania abstrakcyjnego *modelu kształtów* jak i wygenerowania nie abstrakcyjnej reprezentacji graficznej *diagramu kształtów.*

Jak wspomniano, reguła klasycznej *gramatyki kształtu* interpretowana w rozwiązaniach inżynieryjnych składa się z następujących komponentów:

- LHS (Left Hand Side) określający „wzorzec kształtu”, który ulegnie wymianie,
- operator transformacji kształtu (najczęściej =>),
- RHS (Right Hand Side) określający „wzorzec kształtu”, który jest konsekwencją dokonania wymiany.

Na niskim poziomie abstrakcji reguła pozostanie bez zmian. Zalety podjęcia takiej decyzji zostały zreferowane. Jednak w przypadku konieczności przetwarzania modelu wielu *kształtów* sklasyfikowanych, co postulowane jest w założeniach modelu *K-GK*, wzorzec reguły *kształtu* także jest dalece niewystarczający. W pierwszej kolejności – uniemożliwia on wykorzystanie informacji o *klasach* przetwarzanych *kształtów*. Po drugie – w procesie dopasowania reguły (czyli ustalenia faktu istnienia cechy jej stosowalności do danego *kształtu*) wymaga użycia operatorów działających bezpośrednio na reprezentacji graficznej (czasem niemożliwej do przetworzenia akceptowalną transformacją). W dalszej kolejności – z jego użyciem nie będzie możliwe zastosowanie jakichkolwiek zaawansowanych transformacji,

przekształcających obiekty diagramu (*kształty*) podczas zastosowania reguły. Byłaby możliwa tylko zwykła zamiana zadanych symbolicznie *kształtów* (podstawienie).

Konieczne wydaje się zatem opracowanie nowej postaci *reguły kształtu* dla *K-GK*. Reguła taka powinna (jak poprzednio) dokonać przeprowadzenia pewnego komponentu przetwarzanego diagramu (pewnego *kształtu*) w inny – jednak z uwzględnieniem dodatkowych uwarunkowań. I tu reguła tak powinna:

- stosować wytypowaną swoją treścią transformację, której wywołanie jest dodatkowo parametryzowane indeksowanymi wartościami argumentów,
- uniezależnić proces przetwarzania gramatyką od systemu kodowania geometrycznej lokalizacji *kształtów* w euklidesowej przestrzeni n-wymiarowej, jednocześnie utrzymując kontrolę poprawności tej lokalizacji (powinna to być przesłanka stosowalności tej reguły wobec przetwarzanego materiału),
- uniezależnić przyszłe implementacje procesu przetwarzania gramatyką od platformy systemowej.

5.4.3 Przetwarzanie z użyciem abstrakcyjnych klas kształtów

Klasa abstrakcyjna to dodatkowe pojęcie usprawniające procesy przetwarzania *modelu kształtów*.

Definicja 5.6:

> *Klasą abstrakcyjną* w K-GK będziemy nazywali *klasę kształtu*, która nie posiada swojej gramatyki kształtu. ■

Abstrakcyjne *kształty graficzne* mają wiele zalet. Nie będą posiadały swoich odpowiedników *przestrzeni projekcji*, będącej wynikiem prowadzenia procesów z użyciem *K-GK* (nie posiadają reprezentacji graficznych). Powody zastosowania klasy abstrakcyjnej w gramatyce są następujące:

- wprowadzenie abstrakcji kształtu, z możliwością wyrażenia przy jej użyciu elementów początkowego *modelu kształtów*,

- możliwość posłużenia się w gramatyce kształtem nieukończonym (abstrakcyjnym), wspólnym dla wielu ścieżek przetwarzania i każdorazowo wymagającym dalszego rozwijania,
- możliwość wykorzystania dodatkowych relacji uzależniających *klasy kształtów* od siebie (na przykład wyprowadzoną i już nie-abstrakcyjną od jej abstrakcyjnej nadklasy), lecz nie mających związku z reprezentacją graficzną *kształtów* z nimi powiązanych,
- możliwość użycia obiektu, którego występowanie w końcowym wyniku transformacji jest zabronione (musi zostać usunięty / przekształcony przed zakończeniem przetwarzania) – co umożliwia wywarcie dodatkowego wpływu na ścieżkę przetwarzania.

Klasa abstrakcyjna teoretycznie może posiadać *gramatykę kształtu* – przekształcającą kształty w ich interpretacji uogólnionej (przekształcając artybuty dodatkowe opisujące reprezentację graficzną, ale nie przekształcając np. możliwych do wyrenderowania siatkek wielokątów). Obiekt takiej klasy nie będzie zatem posiadał reprezentaji graficznej, możliwej do wyrenderowania w przestrzeni projekcji i musi być dalej przetworzony.

5.4.4 Relacja typu gen-spec pomiędzy klasami kształtów

Po analizie wielu implementacji *gramatyk kształtu* można dojść do wniosku, iż proces przetwarzający z użyciem *gramatyki kształtu* najbardziej chaotycznie zachowuje się w początkowej fazie funkcjonowania. *Diagram kształtów* składa się wówczas z kilku prostych elementów. Celem przetwarzania jest wtedy gównie rozbudowa, a nie redukcja tego modelu. Proste *kształty graficzne* mogą zostać dopasowane do dużo większej ilości reguł łatwiej spełniając kryteria stawiane przez ich LHS. Tym samy decyzja o odpaleniu danej reguły może być podejmowana w sposób bardziej nie deterministyczny. W większości istniejących rozwiązań tylko parametryzacja na poziomie rozpoznawania *kształtów* reguluje zachowanie procedur dopasowujących kształty do LHS. W konsekwencji z uwagi na istnienie jedynie *kształtów* początkowych lub zbliżonych do początkowych owe niedeterministyczne decyzje o odpaleniu reguł poprowadzą ścieżki analizy do zupełnie innych wyników końcowych (różniących się od siebie nawet w zakresie struktury *modelu kształtów*, o detalach reprezentacji graficznych *kształtów* nie wspominając). Przy rozwijaniu *kształtu graficznego* istotne jest przede

wszystkim posiadanie możliwości utrzymania procesu transformacji z użyciem gramatyki pod pełną kontrolą. Eliminuje to jego niedeterministyczny przebieg i zbliża go do osiągnięcia założonych celów.

Sensowne jest wypracowanie bazującego na modelu abstrakcyjnym mechanizmu nadzorującego proces przekształcania *kształtów*. Jak ustalono w koncepcji ogólnej koncepcja K-GK zakłada zaangażowanie hierarchicznego modelu *klas kształtów*, który umożliwi wprowadzenie relacji pomiędzy przetwarzanymi obiektami na najwyższym poziomie abstrakcji. Model ten, posiadając topologię drzewa, będzie analogiczny do standartowego modelu klas znanych choćby ze specyfikacji języków programowania zorientowanych obiektowo.

Kompletny zbiór relacji pomiędzy klasami będzie stanowić podstawę do funkcjonowania procedur sterujących przetwarzaniem *modelu kształtów* w *K-GK*. Hierarchizacja pozwoli na przenoszenie cech *kształtów graficznych* do nadklas lub klas–potomków bez powielania ich definicji. Utrwalona zostanie tym samym informacja wykreowana na podstawie typu lub przeznaczania samego obiektu, który jest reprezentowany przez *kształt graficzny*. Korekty *kształtu* mogą być definiowane w ramach danej *klasy kształtu*. *Klasy*, do których należą *kształty*, będą wpływały na korekty wizualne *kształtu*. Dzięki wykorzystaniu modelu *klas kształtów* możliwe jest produkowanie obiektu bardziej ogólnego (z użyciem reguł nadklasy) lub obiektów wyspecjalizowanych (z użyciem *klasy* wyspecjalizowanej).

5.4.5 Relacje asocjacji kształtów

Ten rodzaj relacji obrazuje sytuację, kiedy *kształty* są wymaganymi komponentami tego samego *modelu kształtu* lub rozwijanie diagramu wymaga ich wzajemnej interakcji. Szczególnym przypadkiem asocjacji może być relacja zawierania (określana często mianem całość-część). Wiedza na temat występowania relacji asocjacji nie będzie wymagała osobnego modelu – relacje te dotyczą konkretnych *kształtów graficznych*, nie klas. Ich zamieszczenie w modelu automatycznie umożliwi instytucja tzw. reguły substytucji, opisanej szczegółowo w następnym podrozdziale.

5.4.6 Reguły substytucji

Reguła substytucji jest elementem jednego ze zbiorów przyszłej gramatyki. Jej użycie w projektowanej gramatyce K-GK będzie skutkowało dodaniem, usunięciem lub konwersją

kształtów. Reguła substytucji posiada, podobnie jak klasyczne reguły kształtu, LHS oraz RHS. Dodatkowo jednak zawiera:

- operującą w n-wymiarowej przestrzeni Euklidesa funkcję – nazwaną *funkcją konwertera*. *Funkcja* k*onwertera* modyfikuje zawartość zbiorów P wszystkich *kształtów*, które otrzymał za pośrednictwem swoich parametrów,
- operującą w n-wymiarowej przestrzeni Euklidesa funkcję zwracającą ustalony stan logiczny – nazwaną *funkcją walidatora*. *Funkcja walidatora* na podstawie zawartości zbiorów P wszystkich *kształtów*, które otrzymała, będzie określała czy materiał kwalifikuje regułę substytucji do uruchomienia, czy też nie (będzie wyrażać metrykę dopasowania tego zbioru do LHS reguły). Tym samym *funkcja walidatora* będzie określała dodatkowy warunek *stosowalności reguły substytucji* wobec *diagramu kształtów*, rozszerzając klasyczną definicję *stosowalności reguły kształtu* gramatyki kształtu.

Definicja 5.7:

Ogólna postać reguły substytucji to:

$$R_S = \{ A \rightarrow B, w(A), k(A,B) \mid A \subset K, B \subset K \}$$

gdzie

$\rightarrow$ jest operatorem reguły substytucji,

K jest zbiorem *kształtów*,

k(A,B) jest *funkcją konwertera*,

w(A) jest *funkcją walidatora*. ■

Można wyróżnić cztery warianty *reguł substytucji*, otrzymywane poprzez zastosowanie różnych operatorów:

- Reguła redukcji,
- Reguła powielania,
- Reguła konwersji,
- Reguła połączenia.

Reguła redukcji

Definicja 5.8:

Regułą redukcji będziemy nazywać wyrażenie:

$$R_R = \{ \alpha\gamma\beta \text{ –=>} \alpha\beta, w(\alpha, \beta, \gamma), k(\alpha, \beta, \gamma) \mid \gamma, \alpha, \beta \subset K \}$$

gdzie

- symbol –=> jest *operatorem* reguły redukcji,

- γ jest usuwaną uporządkowaną listą *kształtów nie terminalnych,*

- α, β są uporządkowanymi listami *kształtów terminalnych* lub *nie terminalnych,*

- $k(\alpha, \beta, \gamma)$ jest *funkcją konwertera* modyfikującym zbiory P *kształtów* w α i β na podstawie *kształtów* z α, β, γ,

- $w(\alpha, \beta, \gamma)$ jest *funkcją walidatora* zawartości zbiorów P *kształtów* z α, β i γ. ■

Wnioski i zalety płynące z tak sformułowanej definicji:

- *kształtów terminalnych* nie można usuwać,
- *konwerter reguły redukcji* umożliwia wprowadzenie modyfikacji *kształtów*, których obecność w *modelu kształtów* spowodowała redukcję,
- *funkcja walidatora* reguły redukcji umożliwia zdefiniowanie dodatkowych warunków wzajemnego usytuowania *kształtów*, będących kryterium do eliminacji *kształtu graficznego* z *modelu kształtów*.

Reguła powielania

Definicja 5.9:

Regułą powielania będziemy nazywać wyrażenie:

$$R_P = \{ \alpha\beta \text{ +=>} \alpha\gamma\beta, w(\alpha, \beta) , k(\gamma, \alpha, \beta) \mid \gamma, \alpha, \beta \subset K \}$$

gdzie

- symbol +=> jest *operatorem* reguły powielania,

- α, β, γ są uporządkowanymi listami *kształtów terminalnych* lub *nie terminalnych,*

- $k(\gamma, \alpha, \beta)$ jest *funkcją konwertera* generującym zbiór P *kształtów* ze nowego zbioru γ na podstawie *kształtów* z α, β,

- $w(\alpha, \beta)$ jest *funkcją walidatora* zawartości zbiorów P *kształtów* z α i β. ■

Wnioski i zalety płynące z tak sformułowanej definicji:

- *kształt dodany* może być od razu *kształtem* terminalnym,
- dodanie *kształtu* nie wprowadzi żadnych zmian w ewentualnej prezentacji graficznej przetwarzanego *modelu kształtów* (*kształt* jest już zlokalizowany w przestrzeni Euklidesa ale nie ma jeszcze żadnej formy). Dopiero rozwinięcie go przy użyciu *gramatyki kształtu* z jego *klasy kształtu* spowoduje wygenerowanie tych danych,
- *funkcja konwertera reguły powielania* umożliwia definiuje usytuowanie nowo utworzonego *kształtu* w przestrzeni Euklidesa – czyniąc to na podstawie *kształtów* z *modelu kształtów*, których obecność spowodowała utworzenie nowego elementu,
- *funkcja walidatora* reguły powielania umożliwia zdefiniowanie dodatkowych warunków wzajemnego usytuowania *kształtów*, przy spełnieniu których w *modelu kształtów* może powstać kolejny *kształt graficzn*y.

Reguła konwersji

Definicja 5.10:

Regułą konwersji będziemy nazywać wyrażenie:

$$R_K = \{ \alpha\lambda\beta :=> \alpha\gamma\beta, w(\alpha, \lambda, \beta), k(\gamma, \alpha, \beta) \mid \gamma, \alpha, \beta \subset K \}$$

gdzie

- symbol :=> jest *operatorem* reguły konwersji,

- α, β, γ są uporządkowanymi listami *kształtów terminalnych* lub *nie terminalnych,*

- k(γ, α, β) jest *funkcją konwertera* modyfikującym zbiór P *kształtów* ze zbioru γ na podstawie *kształtów* z α, β,

- w(α, λ, β) jest *funkcją walidatora* zawartości zbiorów P *kształtów* z α, β, λ. ■

Wnioski z tak sformułowanej definicji:

- *kształt* pows*tały* w wyniku konwersji może być *kształtem* terminalnym
- *funkcja konwertera reguły konwersji* umożliwia korektę usytuowania *kształtu* w przestrzeni Euklidesa – czyniąc to na podstawie *kształtów graficznych* z *modelu kształtów*.
- *funkcja walidatora reguły konwersji* umożliwia zdefiniowanie dodatkowych warunków wzajemnego usytuowania *kształtów*, przy spełnieniu których w *modelu kształtów* może zostać dokonana konwersja *kształtu graficzn*ego.
- *funkcje walidatora* i *konwertera* w regule konwersji będą miały szczególne znaczenie przy tworzeniu systemów symulacyjnych, gdzie przetwarzanie *kształtu graficznego* będzie się często sprowadzało jedynie do korekty jego usytuowania (realizowanej przez *konwerter*) - w sytuacji, gdy *diagram kształtów* zawiera zbiór *kształtów graficznych* dopasowane do *reguły konwersji* i spełniających warunki usytuowania w przestrzeni Euklidesa zadane *funkcją walidatora*.

Reguła połączenia

Definicja 5.11:

Regułą połączenia będziemy nazywać wyrażenie:

$$R_K = \{ \alpha\lambda\beta \text{ *=> } \gamma, w(\lambda, \alpha, \beta) , k(\gamma, \alpha, \beta, \lambda) \mid \alpha, \beta, \lambda \subset K, \gamma \in K \}$$

gdzie

- symbol *=> jest *operatorem* reguły konwersji,

- α, β, λ są uporządkowanymi listami *kształtów nie terminalnych,*

- γ jest symbolem *terminalnym* lub *nietermnalnym,*

- k(γ, α, β, λ) jest *funkcją konwertera* modyfikującym zbiór P *kształtu* γ na podstawie *kształtów* z α, β, λ,

- w(λ, α, β) jest *funkcją walidatora* zawartości zbiorów P *kształtów* z λ, α, β. ■

Wnioski i zalety płynące z tak sformułowanej definicji:

- *reguła połączenia* generuje tylko jeden *kształt*,
- reprezentacje graficzne *kształtów*, ewentualnie wygenerowane już przez *gramatyki kształtów*, są łączone w nowym *kształcie*,
- *funkcja walidatora reguły połączenia* umożliwia zdefiniowanie dodatkowych warunków wzajemnego usytuowania *kształtów*, przy spełnieniu których na *diagramie kształtów* może zostać dokonane połączenie *kształtów graficzn*ych.

Każda z *reguł substytucji* może produkować dowolne *kształty graficzne* (określone w RHS) dokonując także konwersji *kształtu graficznego* do postaci terminalnej. Jest to wyrażane przez zastosowanie operatora „~" przed symbolem identyfikującym *kształt graficzny* w RHS *reguły substytucji*. Wszystkie *kształty* znajdujące się po stronie RHS *reguł substytucji* mogą być *kształtami terminalnymi* lub *nie terminalnymi*

5.4.7 Relacja typu „całość-część" a reguły substytucji

Jak wiadomo *reguły substytucji* będą w praktyce wykorzystywane także do wytwarzania *kształtów*, będących komponentami kolejnych. Dlatego na podstawie treści tych reguł będzie możliwe budowanie diagramów, reprezentujących relacje zawierania typu całość-część (whole-part) występujące pomiędzy *kształtami*. Właściwość ta stanowi cechę systemu umożliwiającego projektowanie hierarchiczne.

5.4.8 Definicja gramatyki K-GK

Definicja 5.12:

Kaskadowa gramatyka kształtu (K-GK) jest czwórką:

$$K_{GS} = < C, K, R, D >$$

gdzie:

C - zbiór *klas kształtów*,

K- zbiór par $\{k_{gen}, k_{spec}\}$, $k_{gen}, k_{spec} \in C$ definiujących występowanie relacji gen-spec pomiędzy *klasami*, przy czym dla każdego k_{gen} istnieje tylko jedno k_{spec} takie, że $\{k_{gen}, k_{spec}\} \in K$ (wykluczone jest wielo-dziedziczenie *klas kształtu*),

R – zbiór *reguł substytucji*,

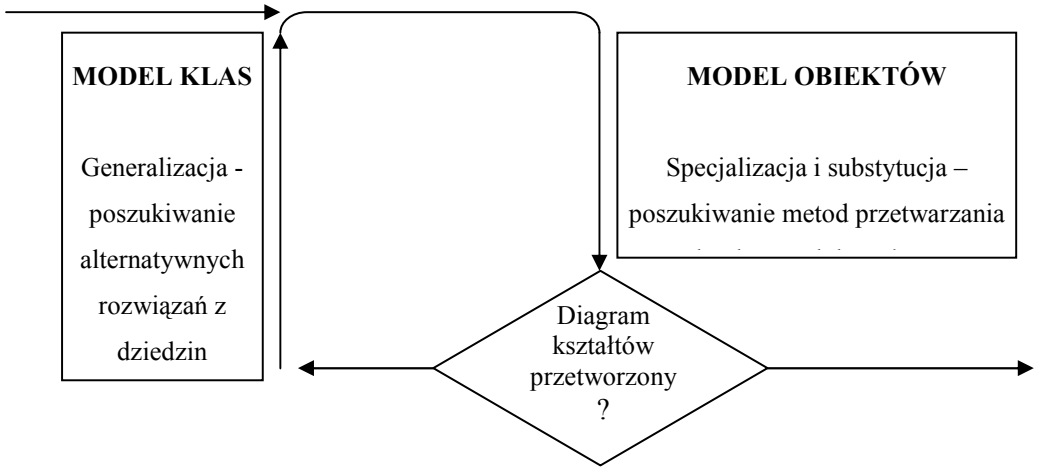
D – lista *kształtów* wraz z ich meta-danymi, będących *kształtami* początkowymi. Będziemy ją nazywać *początkowym diagramem kształtów*. Lista ta jest modyfikowana w toku przetwarzania gramatyką. ■

Definicja 5.13:

Lista D K-GK z chwilą rozpoczęcia wykonywania na niej operacji staje się *diagramem kształtów*.

5.4.9 Interpretacja znaczenia komponentów K-GK

Model *klas kształtów* umożliwia określenie relacji generalizacji i specjalizacji pomiędzy *klasami kształtów*. Powiązania te umożliwiają podjęcie bardziej ogólnej drogi przetwarzania, jeśli nie jest znany sposób wytworzenia obiektu wyspecjalizowanego (skorzystanie z generalizacji). Pozwoli to na obranie „podobnej" ścieżki przetwarzania, zdefiniowanej dla „podobnego" obiektu – czyli wyprowadzonego z tej samej nadklasy *kształtu* (rysunek 5.1).



Rysunek 5.1: Schemat ideowy iteracji prowadzonej podczas procesu przetwarzania *diagramu kształtów* z użyciem gramatyki K-GK .

Model zawierania (whole-part), definiuje relacje pomiędzy *kształtami*, wyrażone za pomocą *reguł substytucji*. Pozwala on na określenie, z jakich elementów i w jakiej ilości powinien składać się dany obiekt, jak istnienie danego obiektu będzie wpływać na jego otoczenie oraz jak czynniki otoczenia będą modyfikować ten obiekt. Weźmy elementarnie prosty przykład z rysunku 5.2. Zaczynając przetwarzanie od obiektu „stołu" możemy rozwinąć go do „stołu z dwoma krzesłami" definiując na bazie funkcjonalnosci niższej warstwy abstrakcji cechy geometryczne tych obiektów:

```
Stol +=> Stol+Krzeslo+Krzeslo, walidate1(Stol),
dodanieDrugiegoKrzesla(OUT Krzeslo, OUT Krzeslo, INOUT Stol);
```

Gdy „stół" posiada dwa „krzesła", będzie wariantem „stołu z parasolem":

```
Stol+Krzeslo+Krzeslo :=> StolZParasolem+Krzeslo+Krzeslo,
walidate2(Stol, Krzeslo, Krzeslo),

przerobkaStolu(OUT StolZParasolem , INOUT Krzeslo, INOUT
Krzeslo, IN Stol);
```

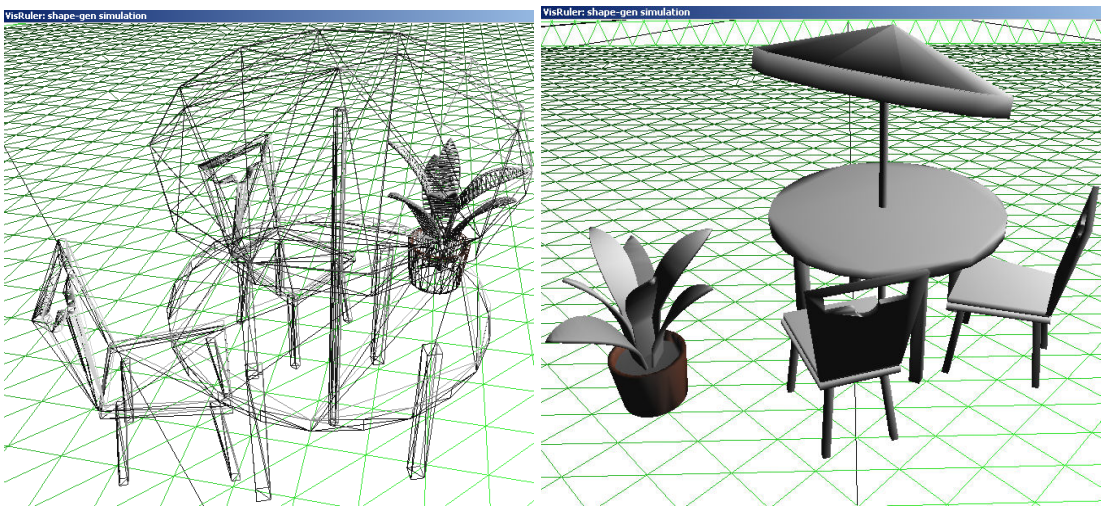
Gdy mamy do czynienia ze „stołem z parasolem", dokładamy „kwiatek":

```
StolZParasolem :=> StolZParasolem+Kwiatek,
walidate3(StolZParasolem), dodajKwiatek(OUT Kwiatek, INOUT
StolZParasolem);
```

Potem łączymy krzesła i stół w jeden obiekt – celem przekształcenia go przy użyciu wspólnej gramatyki kształtu:

```
StolZParasolem + Krzeslo+Krzeslo *=> Zestaw,
walidate4(StolZParasolem,Krzeslo,Krzeslo), stworzZestaw(OUT
Zestaw, IN StolZParasolem, IN Krzeslo, IN Krzeslo);
```

Gdy wyrenderujemy zawartość *diagramu kształtów* (zakładając, iż reprezentacje graficzne poszczególnych *kształtów* to modele złożone na przykłaz z siatek wielokątów) to otrzymamy wynik prezentowany na rysunku 5.2. Zakładamy naturalnie, iż funkcje operujące na geometrycznym ustyuowaniu obiektów zostały zdefiniowane.



Rysunek 5.2: Reprezentacja graficzna *diagramu kształtów* po wykonaniu *reguł substytucji* z przykładu (źródło własne).

Geometryczne usytuowanie obiektów znajdujących się na *diagramie kształtów* to drugi cel przetwarzania z wykorzystaniem zbioru *reguł substytucji*. Informacja ta (w tym ta określająca usytuowanie obiektu w przestrzeni projekcji) musi także zostać przetworzona. Daltego każda z *reguł substytucji* posiadać może operującą w przestrzeni Euklidesa funkcję przekształcającą parametry określające faktyczne usytuowanie *kształtu*. Określenie tej cechy mianem usytuowania lub rozlokowania *kształtów* jest tutaj najbardziej adekwatne, gdyż specyfikacja struktury meta-danych określających geometryczne własciwości *kształtu* jest otwarta. W większości przypadków zastosowań rozwiązania w mata-danych zawarte zostaną takie informacje jak pozycja, rotacja czy wartości współczynników przeskalowania *kształtu*. Nie jest to jednak ściśle narzucone. Wartości w meta-danych zadane są zbiorem identyfikowalnych n-tek, przy czym *n* to wymiar przestrzeni Euklidesa, w której osadzone są *kształty graficzne*.

5.4.10 Algorytmy przetwarzania diagramu kształtów

Po zdefiniowaniu statycznego modelu gramatyki możliwe jest rozpoczęcie prac nad ustabilizowaniem algorytmów sterujacych przetwarzaniem gramatyką. Rozważania należy rozpocząć od głównego algorytmu sterującego całym procesem przetwarzania *diagramu kształtów*. Modyfikacja przetwarzanego materiału na pewno będzie prowadzona wieloetapowo. W kolejnych krokach rozważań należy wyspecyfikować procedurę określającą proces generalizacji *kształtu graficznego*, uruchamiany gdy konieczne staje się odszukanie nowej *klasy*

kształtu w bieżącej wiedzy lub wiedzy pochodzącej z innej dziedziny. Następnie – algorytm sterujący przetwarzaniem *diagramu kształtów regułami substytucji* i wreszcie – algorytm sterujący procesem dopasowania LHS konkretnej *reguły kształtu* do aktualnej zawartości *diagramu kształtów*.

Główna procedura sterująca

Procedura przetwarzania *diagramu kształtów* gramatyką K-GK otrzymuje zbiór *kształtów początkowych*. Przetwarzanie jest realizowane z wykorzystaniem:

- modelu *klas kształtów* dla danej *dziedziny*,
- zbioru *reguł substytucji*,
- *diagramu kształtów*.

Celem głównym jest eliminacja *kształtów nie terminalnych* z *diagramu kształtów.*

Procedura będzie zakładała istnienie wielu etapów przetwarzania *diagramu kształtów,* nazywanych *iteracjami*. Prowadzone będzie kontrolowane uruchamianie algorytmu sterującego konkretną iteracją, oraz (w określonych sytuacjach) algorytmu poszukiwania nowych *klas kształów*. Procedura kontrolująca uruchamianie reguł będzie wykonywała kod sterujący wyrażony za pośrednictwem dodatkowego diagramu, zwanego *diagramem sterującym*. Diagram ten określi kolejność naprzemiennego uruchamiania iteracji i prowadzenia procesu generalizacji. Diagram można będzie wyrazić w postaci graficznej, umieszczając na nim graficzne symbole rozkazów wykonania iteracji, generalizacji, wykonania skoków, powtórzeń wielkrotnych itd. Diagram ponadto musi określać krok rozpoczynający i kończący przetwarzanie. Tak zdefiniowany *diagram sterujący* będzie można wprowadzać do interfejsów użytkownika w gotowych systemach tworzonych na podstawie rozważanej metodyki, udostępniając narzędzia do jego edycji.

Zakończenie wykonania procedury odbędzie się z chwilą osiągnięcia ostatniego kroku naniesionego na *diagram sterujący* lub po stwierdzeniu braku jakichkolwiek *kształtów* nie terminalnych przez *algorytm sterujący iteracją*. Warto dodatkowo zauważyć, iż w przypadku użycia proponowanego modelu do wsparcia procesów sterowania i symulacji w grafice (przetwarzanie ciągłej) zakończenie wykonania algorytmu można będzie uzależniać od warstw wyższych, sterujących takimi procesami (a tym samym na przykład od użytkownika).

Algorytm sterujący iteracją

Wyspecyfikowanie tego algorytmu jest kluczowe dla określenia reguł prowadzenia konkretnego procesu transformacji w *diagramie kształtów*. Wobec mnogości możliwości postępowania pewne regulacje zwiazane z projektowaniem algorytmu trzeba było przyjmować aksjomatycznie. Inne założenia zostały ustabilizowane w konsekwencji liczych prób (także testów implmentacji) i ewaluacji pod kątem możliwości wykorzystania w konkretnych zagadnieniach praktycznych.

Treść algorytmu będzie następująca:

1. *Oznacz wszystkie kształty z diagramu kształtów jako nie wykorzystane w bierzącej iteracji,*
2. *Sprawdź czy w diagramie kształtów istnieją kształty nie terminalne. Jeśli nie –* ***zakończ*** *zwracając tą informację,*
3. *Weź pierwszy element r ze zbioru R reguł substytucji* K-GK *wykonaj:*
   3a. Uruchom algorytm dopasowania reguły kształtu r (opisany w następnym podrozdziale),

   3b. Gdy otrzymano wynik negatywny (nie nastąpiło dopasowanie) przejdź do 3f,

   3c. Wykonaj przekształcenie diagramów kształtów dopasowaną regułą substytucji r podstawiając do LHS tej reguły kombinację kształtów v_i otrzymaną za pośrednictwem algorytmu dopasowania,

   3d. Uruchom funkcję konwertera reguły substytucji r podstawiając do niej kształty z diagramu kształtów zgodnie z treścią LHS i RHS reguły substytucji,

   3e. Gdy po wykonaniu reguły substytucji powstały kształty terminalne – przekształć je gramatykami kształtu zawartymi w ich klasach kształtu,

   3f. Weź kolejny element ze zbioru R. Jeśli jest - przejdź do 3a,
4. *Sprawdź, czy na diagramie kształtów isteniejąjakiekolwiek kszałty oznaczone przez algorytm dopasowania reguł substytucji jako wykorzystane. Jeśli nie –* ***zakończ*** *z*

*informacją o konieczności prowadzenia generalizacji klas kształtów, jeśli tak – **zakończ** normalnie.*

Algorytm będzie uruchamiany wielokrotnie. Operując w ramach jednej iteracji przeprowadzi walidację *reguł substytucji* pod kątem możliwości ich wykorzystania. Będzie przestrzegał uporządkowania reguł na ich liście. Każdy z *kształtów graficzncych*, który został już przetworzony w bieżącej iteracji nie będzie mógł brać udziału w dalszym procesie dopasowania.

Tym samym algorytm dopuszcza w każdej iteracji możliwość **jednokrotnej** aplikacji zmiany jakościowej *kształtu* lub możliwość **jednokrotnego** wywarcia wpływu na jakościową zmianę innego *kształtu* w *diagramie kształtów*.

Algorytm dopasowania *reguły substytucji*

Algorytm ten był wielokrotnie modyfikowany ewoluując w czasie trwania badań nad procedurami najbardziej optymalnego wykorzystania wiedzy zawartej w proponowanym modelu. Jego finalna postać jest następująca:

1. *Utwórz zbiór K wszystkich klas kształtu, jakich elementy znajdują się w LHS reguły kształtu,*
2. *Stwórz zbiór R referencji do wszystkich kształtów z diagramu kształtów, których klasy kształtu są równe któremukolwiek elementowi K. Ignoruj elementy już wykorzystane w bieżącej iteracji. Diagram kształtów przeglądaj sekwencyjnie (zgodnie z kolejnością na liście diagramu kształtów),*
3. *Zakładając, iż liczba kształtów (nie liczba klas kształtów) w LHS rozważanej regule substytucji wynosi n stwórz zbiór V n-elementowych wariacji bez powtórzeń zbioru R dla każdej z klas. Wyklucz ze zbioru V te elementy, w których na dowolnej pozycji klasa kształtu elementu z LHS reguły substytucji nie pasuje do klasy kształtu odpowiadającego kształtu w elemencie zbioru V,*
4. *Wytypuj pierwszy element $v_i \in V$,*

   4a. Sprawdź, zależnie od rodzaju reguły kształtu czy dla wytypowanego elementu v_i nie definiuje ona transformacji modyfikującej którykolwiek z ewentualnych kształtów terminalnych zawartych w v_i. Jeśli tak – przejdź do 4d,

4b. Wywołaj funkcję walidatowa $w(v_i)$,

*4c. Gdy funkcja $w(v_i)$ zwróci prawdę oznakuj kształty z R jako wykorzystane w bierzącej iteracji i **zakończ** zgłaszajac dopasowanie, oraz zwracając v_i,*

4d. Weź kolejny element $v_i \in V$ i przejdź do 4a,

5. ***Zakończ** zgłaszając brak dopasowania.*

Algorytm umożliwi odnalezienie pierwszego, a potem kolejnych dopasowań *reguły substytucji* eliminując takie *kształty*, które nie mogą już podlegać zmianom.

Procedura poszukiwania klas kształtów

Ogólna, określona przez omawianą metodykę zasada brzmi, iż model klas K-GK będzie stosowany do realizowania procesu *generalizowania kształtów*. Oznacza to, iż z uwagi na możliwość istnienia nadklasy względem *klasy kształtu* wykorzystanie tej nadklasy wesprze transformację *kształtu graficznego* - gdy ta nie może być prowadzona jedynie w oparciu o opis klasy wyprowadzonej. Inna zaleta uwidacznia się poprzez fakt, iż generalizacja może zostać wymuszona poprzez interpretację *diagramu sterującego*. To drugie rozwiązanie daje możliwość wyprowadzenia procesu decycyjnego do interfejsu użytkownika systemu przetwarzajacego, a tym samym przejęcie przez niego kontroli nad etapami całego procesu. Wejdzie on wtedy w interaktywny tryb pracy prowadzony z narzędziem generującym. Manualnie wymuszana generalizacja klas będzie mogła zostać przeprowadzona na żądanie, stymulując poszukiwania innych niż znane użytkownikowi, alternatywnych rozwiązań. Przetwarzany *kształt* będzie wówczas interpretowany jako bardziej ogólny, więc w kolejnym kroku transformacji zostaną (być może, zależnie od kształtu wiedzy) zastosowane wobec niego inne, bardziej ogólne metody obróbki. W konsekwencji generalizacji może on znów zostać wyspecjalizowany (po kolejnej iteracji procesu przetwarzania) w zupełnie inny *kształt* niż ten, którym był poprzednio. Operacja specjalizowania *kształtów* jest realizowania za pośrednictwem *reguł substytucji*.

Poszukiwanie nadklasy będzie polegało na wyodrębnieniu jej ze zbioru klas na podstawie relacji gen-spec między *klasami kształtów* K-GK.

W przypadku niepowodzenia procedury wyszukującej nadlklasy można rozważyć skorzystanie z pomocy interfejsu, który dostarczy odpowiednik znaczeniowy *klasy kształtu* występujący w innej gramatyce. *Klasa kształtu* wraz z procedurami jego modyfikacji może być zdefiniowana w innej bazie wiedzy i pod inną leksykalnie wyrażoną nazwą. Zastosowanie

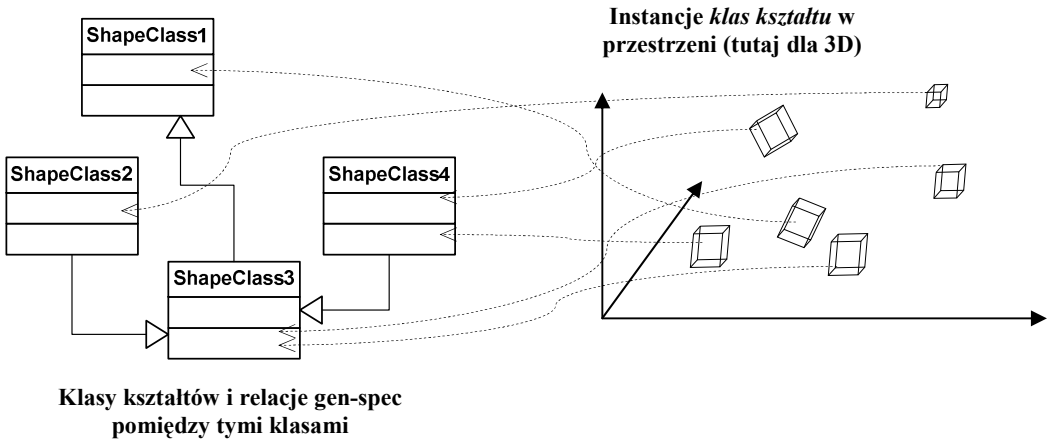
rozwiązania umożliwiającego dostarczenie odpowiedników ontologicznych nazw *klas kształtów* umożliwi wykorzystanie wiedzy kolektywnej - pochodzącej od różnych ekspertów, lecz określonej nad tą samą dziedziną. Interfejs umożliwiający wymianę takiej wiedzy zostanie przedstawiony w kolejnych podrozdziałach.

5.5 Interpretacja logiki funkcjonowania systemów graficznych opartych o model K-GK

Niniejszy podrozdział podsumowuje teoretyczne rozważania na temat architektury modelu przetwarzania K-GK, podkreślając jego zalety oraz określając znaczenie komponentów rozlokowanych w poszczególnych warstwach. Kasyfikuje także terminologię, wprowadzoną licznie w poprzednich podrozdziałach. Właściwa interpretacja logiki funkcjonowania modelu K-GK będzie podstawą do rozpoczęcia rozważań nad technicznymi aspektami implementacji graficznych systemów generacyjnych wykorzystujących taki model.

5.5.1 Znaczenie reguł substytucji w gramatyce K-GK

Jedną z ważniejszych przesłanek do wprowadzenia wielowarstwowej struktury bazy wiedzy było umożliwienie łatwego dostarczania wiedzy eksperckiej do implementacji modelu K-GK. Wiedza o *klasach kształtów* jest wiedzą uniwersalną i w pełni niezależną od informacji o konkretnych *kształtach graficznych*. Zawiera ona informację, iż przykładowo określony typ *kształtu* jest wariantem wyspecjalizowanym innego typu – bez wchodzenia w jakiekolwiek szczegóły tej specjalizacji (Rysunek 5.3). Jest to wiedza zdefiniowana na najwyższym poziomie abstrakcji ze wszystkich zawartych w modelu.

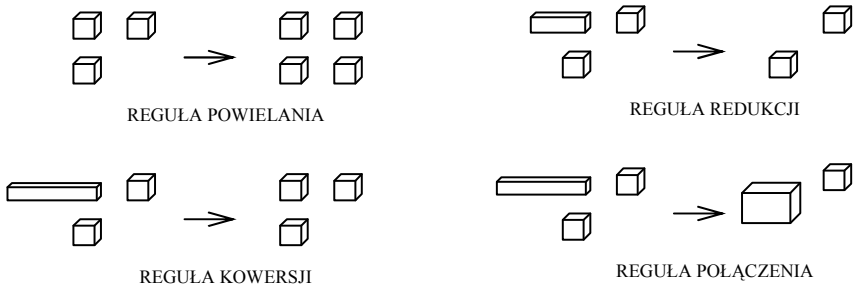


Rysunek 5.3: *Klasy kształtów* w modelu K-GK (źródło własne).

Wiedza o wzajemnym uzależnianiu występowania *kształtów graficznych* konkretnego typu (konkretnej *klasy kształtu*) jest już wiedzą „wykonawczą". Stanowi ona podstawę do:

- dalszej klasyfikacji *kształtów* (konwersji),
- przetwarzania informacji o usytuowaniu *kształtów* w przestrzeni projekcji,
- dodania, usunięcia lub połączenia *kształtu graficznego.*

Modyfikacje te (jak i sama wiedza) są także ujęte abstrakcyjnie, gdyż bazują wyłącznie na rozróżnianiu *klas kształtu* danych obiektów - abstrahując od ich reprezentacji graficznej, która może nawet fizycznie nie istnieć. Przetwarzanie na tym poziomie będzie przykładowo modyfikowało jedynie typ i położenie obiektu w sąsiedztwie innych jeśli zbiór tych innych zawierał przed modyfikacją określone elementy (Rysunek 5.4).



Rysunek 5.4:. Graficzna prezentacja funkcjonowania *reguł substytucji* w różnych wariantach (bez uwzględniania działania funkcji konwerterów w tych regułach) (źródło własne).

Kształty widoczne na powyższym rysunku są renderowane wyłącznie w celach poglądowych . Na obecnym poziome abstrakcji nie posiadają jeszcze *reprezentacji graficznej*. Ich reprezentację można co najwyżej uogólnić do punktu (wierzchołka) umieszczonego w n-wymiarowej przestrzeni Euklidesa. *Funkcje konwerterów* wykorzystują informację o usytuowaniu *kształtów* w celu wygenerowania nowej postaci tej informacji (opisującej *kształty* w RHS każdej z *reguł substytucji*).

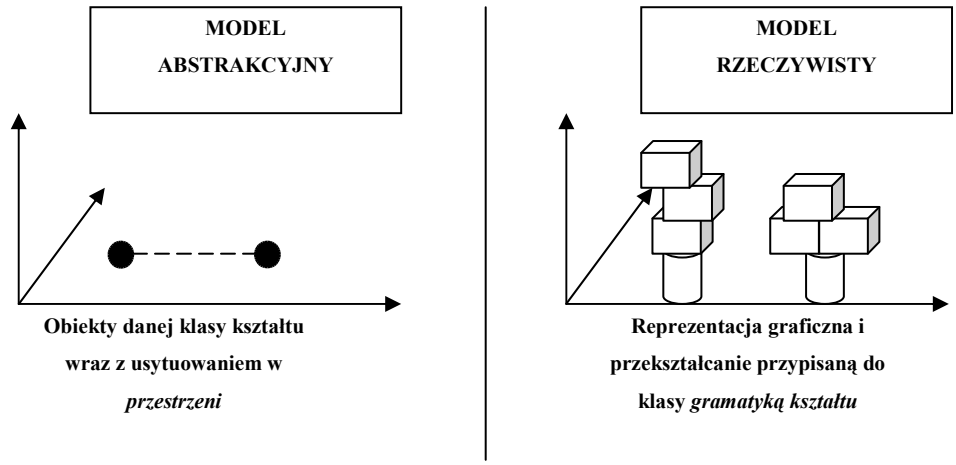
5.5.2 Abstrakcyjna interpretacja klas kształtów – czyli izolacja procesów przetwarzania klas kształtów od reprezentacji graficznej kształtów

Model abstrakcyjny *klas kształtów* zakłada wyizolowanie funkcjonalności przetwarzającej same *klasy* konkretnych *kształtów graficznych* i dane o ich lokalizacji od funkcjonalności umożliwiającej dostarczenie graficznej reprezentacji tych obiektów.

Jak wspomniano - takie podejście umożliwia pełne uniezależnienie procesu przetwarzania *kształtów graficznych* od reprezentacji graficznej. Jak założono wcześniej - operacje na *kształtach* w najwyższej warstwie abstrakcyjnej sprowadzają się wyłącznie do:

- wykonywania konwersji ich *klas* poprzez zastosowanie *reguły konwersji*,
- dodawania lub usuwania *kształtów* w konsekwencji zastosowania *reguły redukcji* lub *reguły powielania*,
- łączenia *kształtów*, z kwalifikowaniem produktu połączenia jako *kształt* danej *klasy* (z użyciem *reguły połączenia*).

Ponadto do wykonywania operacji geometrycznych polegających na przekształceniu struktur opisujących pozycję każdego z *kształtów*. Jedyną informacją wymaganą na tym poziomie abstrakcji jest wymiar przestrzeni Euklidesa, w której opisane transformacje będą miały miejsce (Rysunek 5.5). Tym samym – proces przetwarzania *kształtów* jest transformacją całkowicie uniezależnioną od reprezentacji graficznej samych *kształtów*.



Rysunek 5.5: Abstrakcja *klas kształtów* od ich reprezentacji graficznej (źródło własne).

Model rzeczywisty zakłada przetwarzanie reprezentacji graficznej danego *kształtu graficznego*. Każda *klasa kształtu* zawiera *gramatykę kształtu*, która może być wykorzystana do dalszego przetwarzania sklasyfikowanego i usytuowanego w przestrzeni *kształtu graficznego*.

5.5.3 Uporządkowanie strukturalne przetwarzanej wiedzy

Wiedzę ostatecznie zakwalifikowaną jako komponenty modelu K-GK można także podzielić w następujący sposób: komponenty uzależnione od zadania projektowego przetwarzane przez implementacje wykorzystujące model oraz komponenty uniezależnione. „Uzależnienie" oznacza, iż dany materiał musi zostać przygotowany specjalnie dla każdego nowego zadania, do którego implementacja systemu wykorzystującego model K-GK jest realizowana. Przykładowo – generowanie wielopiętrowego budynku mieszkalnego może bazować na ogólnie przyjętych *klasach kształtu* identyfikujących komponenty budynku, może także wykorzystać ogólnie ujęte relacje gen-spec pomiędzy tymi *klasami*. Nie może natomiast bazować na nie modyfikowanej liście reguł decydujących o koniecznych do przeprowadzania transformacjach. Transformacje te będą bowiem modyfikowały wzajemne ułożenie komponentów budynku. Będą także pośrednio wpływały na podejmowanie decyzji o kolejnych krokach jego rozbudowy (to określa treść *reguł substytucji*). Podobnie, choć w mniejszym stopniu, uzależniona od zadania projektowego jest treść zbiorów *gramatyk kształtu* przypisywanych do konkretnych *klas kształtu* (Tabela 5.1, Tabela 5.2).

Tabela 5.1: Uporządkowanie strukturalne wiedzy przetwarzanej przez K-GK, elementy niezależne od postawionego zadania projektowego.

Element	*Klasy kształtów graficznych*	*Model klas kształtów graficznych*
Reprezentacja / zawartość	Zbiór *klas kształtów*	Zbiór relacji typu gen-spec pomiędzy *klasami kształtów*
Zależny od techniki renderowania *kształtu graficznego*	nie	nie

Tabela 5.2: Uporządkowanie strukturalne wiedzy przetwarzanej w ramach K-GK, elementy uzależnione od postawnionego zadania projektowego (sformułowane w celu prowadzenia przekształceń w celu ozwiązania zadania).

Element	*Model fizyczny klas kształtów*	*Funkcje konwerterów reguł substytucji*	*Gramatyka kształtu*	*Reprezentacja graficzna kształtu*
Reprezentacja / zawartość	Zbiór *reguł substytucji*	Funkcje modyfikujące usytuowanie obiektów przetwarzanych przez *reguły substytucji* w przestrzeni Euklidesa	Czwórka zbiorów, zgodnie z definicją	Metody renderujące, zakodowane w warstwie rysującej
Zależny od techniki renderowania *kształtu graficznego*	nie	nie	pośrednio	tak

Warto zauważyć, że technika renderowania *kształtu graficznego* wpływa na proces przygotowania danych do zadania projektowego jedynie w przypadku reprezentacji graficznej konkretnych *kształtów*. Wprowadzenie poziomów abstrakcji przetwarzania umożliwia rozwiązanie zadania projektowego bez udziału reprezentacji graficznej *kształtów*, tym samym bez konieczności interpretowania wyglądu w przestrzeni projekcji.

5.6 Techniczne aspekty projektowania systemów opartych model K-GK

Nawiązanie do koncepcji użycia modelu K-GK w jakimkolwiek zastosowaniu praktycznym wymaga zaprojektowania modułu przetwarzajacego *kształty graficzne*, działającego na proponowanym wielowarstwowym modelu abstrakcyjnym. Jego zaprojektowanie wymaga:

- określenia podsystemu importującego i przechowującego informacje o *klasach kształtów* i relacjach pomiędzy *klasami kształtów* (model abstrakcyjny najwyższego poziomu),
- określenia modułu dopasowującego *reguły substytucji* i przetwarzającego *diagram kształtów* przy uruchamianiu tych reguł,
- określenia podsystemu zarządzającego *funkcjami konwerterów reguł substytucji, funkcjami walidatorów reguł substytucji* i przetwarzającego dane o usytuowaniu *kształtów graficznych* w wirtualnej przestrzeni Euklidesa (wiedzę sterująca procesem przebudowy *kształtów*),
- stworzenia założeń dla pracy komponentu sterującego całym procesem przetwarzania z użyciem gramatyki K-GK.

Przykładowa implementacja testowego systemu wykorzystującego model K-GK zostanie opisana w następnym podrozdziale. Będzie także stanowić podstawę do prowadzenia eksperymentów dotyczących przetwarzania konkretnych *diagramów kształtów*.

5.6.1 Reprezentacja K-GK z wykorzystaniem XML

XML stał się dziś standardowym językiem wymiany meta-wiedzy i reprezentacji wiedzy. Logiczna struktura budowana przy użyciu znaczników XML umożliwia łatwe zakodowanie uporządkowanych hierarchicznie treści, czemu XML zawdzięcza swą ogromną popularność. Zakodowanie danych w standardzie XML gwarantuje ich przenośność i znacznie ułatwia implementację późniejszych narzędzi kodujących i dekodujących dane na potrzeby systemów przetwarzających wiedzę. Dodatkowe zalety języka niesie możliwości tworzenia reprezentacji graficznych wyrażonej w nim struktury danych oraz otwarty standard umożliwiający rozszerzanie. Język XML może więc stanowić podstawę do opracowania języków pochodnych,

wyspecjalizowanych do opisu wiedzy dla konkretnych rozwiązań inżynieryjnych. Istnieje obecnie wiele takich modyfikacji. Wśród nich choćby RuleML (*Rule-based Markup Language*) [Bol00], będący standardem umożliwiającym hierarchiczny zapis wiedzy opartej na kilku typach reguł.

Przy znacznym stopniu złożoności struktury wiedzy opisującej K-GK mało prawdopodobnym wydaje się znalezienie gotowego rozwiązania, bazującego na XML i umożliwiającego zakodowanie tej wiedzy. Celowe zatem wydaje się opracowanie dedykowanego języka, umożliwiającego reprezentację wiedzy systemów opartych o teorię K-GK.

CSGL (*Cascade Shape Grammar Language*) będzie nowym, rozszerzającym XML językiem zaprojektowanym do wyrażenia wiedzy opisującej K-GK. CSGL umożliwi wyrażenie elementów wiedzy, które będą wykorzystywane do przetwarzania w warstwowym modelu abstrakcyjnym. Takie podejście pozwoli stworzyć rozwiązanie uniwersalne – gdyż ponownie uniezależnione od sposobu reprezentacji graficznej *kształtów* oraz platformy systemowej wspierającej jej przetwarzanie.

5.6.2 Język CSGL

Struktura języka CSGL będzie bazowała na XML, rozszerzając definicje przestrzeni nazw XML. Podstawowy element logiczny utrzymującym informację o strukturze języka CSGL będzie nosił nazwę *csg* (*cascade shape grammar*). Przykład podstawowej definicji prezentuje rysunek 5.6:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<csgl:csg xmlns="http://home.agh.edu.pl/mitu/csgl-namespace">
</csgl:csg>
```

Rysunek 5.6: Element *csg* języka CSGL (źródło własne).

Następujące elementy definiują poszczególne zbiory gramatyki K-GK:

- classes – zbiór *klas kształtów*,
- substitutions – zbiór *reguł substytucji*.

Zatem struktura ogólna dokumentu CSGL będzie posiadała jak na rysunku 5.7:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<csgl:csg xmlns="http://home.agh.edu.pl/mitu/csgl-namespace">
<csg>
 <classes>
       <class>definicja klasy kształtu</class>
 ...
 </classes>
 <substitutions>
       <subrule>definicja reguły substytucji</subrule>
       ...

 </substitutions>
</csg>
</csgl:csg>
```

Rysunek 5.7. Ogólna postać dokumentu CSGL (źródło własne).

Reprezentacja *klasy kształtu*

Każda *klasa kształtu*, wraz ze wszystkimi swoimi własnościami w języku CSGL jest reprezentowana przez znacznik <class> (rysunek 5.8):

```
<class>
 <classname>nazwa klasy kształtu</classname>
 <classid>identyfikator</classid>
 <superclass>nazwa klasy kształtu</superclass>
 <classdescription>opis</classdescription>
 <abstractclass></abstractclass>
 <shapegrammar>
       ( ... )
 </shapegrammar>
</class>
```

Rysunek 5.8: Reprezentacja *klasy kształtu* w języku CSGL (źródło własne).

Na powyższym rysunku:

- znacznik `<superclass>` określa zdefiniowaną przy pomocy <class> klasę nadrzędną,

- znacznik `<shapegrammar>` identyfikuje *gramatykę kształtu* danej *klasy kształtu*,
- istnienie znacznika `<abstractclass>` określa *klasę kształtu* jako abstrakcyjną.

Reprezentacja gramatyki kształtu w klasie kształtu

Każda gramatyka K-GK jest opisywana w języku CSGL jako obiekt abstrakcyjny. Notacja stosowana przy określaniu takiej gramatyki zależy treści materiału przetwarzanego przez zewnętrzne narzędzia. Aby umożliwić izolowanie abstrakcyjnej warstwy gramatyki, język CGSL może dostarczyć jedynie możliwość określania (markowania) gramatyki kształtu unikatowym identyfikatorem, który może następnie być interpretowany jako uchwyt lub wskaźnik do kodu realizującego przewarzanie taką gramatyką w implementacji konkretnego narzędzia. Znacznik `<shapegrammar>` definiuje identyfikator gramatyki wiązanej później z *klasami kształtów*. Jego interpretacja zależy od narzędzi operujących na reprezentacjach graficznych *kształtów* (Rysunek 5.9):

```
<shapegrammar>
 <grammarname>nazwa klasy kształtu</grammarname>
 <grammarreference>identyfikator hashcode</grammarreference>
 <grammardata>
       <grammardataentry> dane </grammardataentry>
       ...
 </grammardata>
</shapegrammar>
```

Rysunek 5.9: Reprezentacja referencji *gramatyki kształtu* w języku CSGL (źródło własne).

Na powyższym rysunku:

- Znacznik `<grammarreference>` określa handler lub wskaźnik do *gramatyki kształtu*, definiowanej i implementowanej w zewnętrznym module/narzędziu,
- Znacznik `<grammareference>` umożliwia wyrażenie dodatkowych meta-danych, określających przeznaczenie gramatyki i przedmioty przetwarzania z jej wykorzystaniem (w rozumieniu uzależnionym od *dziedziny*).

Ogólna reprezentacja reguły substytucji

Każda *reguła substytucji*, wraz ze wszystkimi swoimi własnościami w języku CSGL jest reprezentowana przez znacznik <subrule>, jak pokazano na rysunku 5.10:

```
<subrule type=”typreguly”>
 <ruleid>identyfikator reguły substytucji</ruleid>
 <ruleexplicit></ruleexplicit>

 <rulelhs>
              <classcomponent>
                     <classname>nazwa</classname>
                     <terminator/>
              </classcomponent>
              ...
 </rulelhs>
 <rulerhs>
              <classcomponent>
                     <classname>nazwa</classname>
                     <terminator/>
              </classcomponent>
              ...
 </rulerhs>
 <validator>
       <validatorname>nazwa funkcji walidatora</validatorname>
       <validatorreference>identyfikator hashcode
       </validatorreference>
 </validator>
 <converter>
       <convertername>nazwa funkcji konwertera</convertername>
       <converterreference>identyfikator hashcode
       </converterreference>
 </converter>
</subrule>
```

Rysunek 5.10: Reprezentacja *klasy kształtu* w języku CSGL (źródło własne).

Na rysunku 5.10 znajdują się:

- Znacznik `<lhs>` zawiera definicje LHS *reguły substytucji*. W nim definiowane są znaczniki `<classcomponent>`,
- Znacznik `<rhs>` zawiera definicje RHS *reguły substytucji*. W nim definiowane są znaczniki `<classcomponent>`,
- Znacznik `<ruleexplicit>`,
- Znacznik `<classcomponent>` określa jeden *kształt* danej *klasy kształtu*, znajdujący się w LHS lub RHS reguły. Zawiera:
    - Znacznik `<classname>` określający nazwę *klasy kształtu,*
    - Opcjonalny znacznik `<terminator>`, którego istnienie klasyfikuje *kształt* jako *terminalny.*
- Znacznik `<converter>` określa właściwości funkcji konwertera w *regule substytucji*. Zawiera:
    - Znacznik `<convertername>` określający nazwą funkcji *konwertera*.
- Znacznik `<converterreference>` określa handler lub wskaźnik do *funkcji konwertera*, definiowanej i implementowanej w zewnętrznym module/narzędziu.

Reprezentacja *reguły redukcji*

Reguły redukcji będą identyfikowane przy użyciu atrybutu `type` znacznika `<subrule>` o wartości „`shapereduce`" (Rysunek 5.11)

```
<subrule type="shapereduce">
        { ... }
</subrule>
```

Rysunek 5.11: Reprezentacja *reguły redukcji* w języku CSGL (źródło własne).

Reprezentacja *reguły powielania*

Reguły powielania będą identyfikowane przy użyciu atrybutu `type` znacznika `<subrule>` o wartości „`shapeduplication`" (Rysunek 5.12).

```
<subrule type="shapeduplicate">
        { ... }
```

```
</subrule>
```

Rysunek 5.12. Reprezentacja *reguły powielania* w języku CSGL (źródło własne).

Reprezentacja *reguły konwersji*

Reguły konwersji będą identyfikowane przy użyciu atrybutu `type` znacznika `<subrule>` o wartości „`shapeconvert`" (Rysunek 5.13).

```
<subrule type="shapeconvert">
        { ... }
</subrule>
```

Rysunek 5.13: Reprezentacja *reguły konwersji* w języku CSGL (źródło własne).

Reprezentacja *reguły połączenia*

Reguły konwersji będą identyfikowane przy użyciu atrybutu `type` znacznika `<subrule>` o wartości „`shapemerge`" (Rysunek 5.14).

```
<subrule type="shapemerge">
        { ... }
</subrule>
```

Rysunek 5.14: Reprezentacja *reguły połączenia* w języku CSGL (źródło własne).

5.6.3 Walidacja języka CSGL

Pełna specyfikacja CSGL może zostać wyrażona za pomocą języka DTD (*Document Type Definition*) [W3c08]. Walidator CSGL będzie przy postawionych w podrozdziale założeniach został przedstawiony na rysunku 5.15.

```
<!--
CSGL DTD Validator, Michal Turek
-->
<!ELEMENT csg (classes, substitutions)>
<!ELEMENT classes (class+)>
<!ELEMENT class (classname, superclass+, classdescription+, ab-
stractclass+, shapegrammar+)>
  <!ELEMENT classname (ID)>
  <!ELEMENT classname (#PCDATA)>
  <!ELEMENT classid EMPTY>
```

```
  <!ELEMENT superclass EMPTY>

  <!ELEMENT abstractclass EMPTY>

  <!ELEMENT shapegrammar (grammarname, grammarreference, grammarda-
ta)>

   <!ELEMENT grammarname (ID)>

   <!ELEMENT grammarreference (#PCDATA)>

   <!ELEMENT grammarndata (grammardataentry*)>

    <!ELEMENT grammardataentry (#PCDATA)>

<!ELEMENT substitutions (subrule*)>

 <!ELEMENT subrule (ruleid, ruleexplicit?, rulelhs, rulerhs, conver-
ter)>

  <!ATTLIST subrule type (shapere-
duce|shapeduplicate|shapeconvert|shapemerge)>

  <!ELEMENT ruleid (#ID)>

  <!ELEMENT ruleexplicit EMPTY>

  <!ELEMENT rulelhs (classcomponent+)>

  <!ELEMENT rulerhs (classcomponent+)>

   <!ELEMENT classcomponent (classname, terminator?)>

    <!ELEMENT classname (#ID)>

    <!ELEMENT terminator EMPTY>

  <!ELEMENT converter (convertername, converterreference+)>

   <!ELEMENT convertername (#ID)>

   <!ELEMENT converterreference (#PCDATA)>

  <!ELEMENT validator (validatorname, validatorreference+)>

   <!ELEMENT validatorname (#ID)>

   <!ELEMENT validatorreference (#PCDATA)>
```

Rysunek 5.15: Walidator DTD składni języka CSGL (źródło własne).

Dzięki wprowdzaniu języka CSGL możliwe będzie sprawne projektowanie komponentów systemów wykorzystujących proponowaną metodykę. Język jest łatwo rozszerzalny, co umożliwi projektowanie dodatków składniowych do jego definicji w przypadku, gdy gramatyce K-GK w konkretnej implementacji systemu powinny towarzyszyć jeszcze inne dane. Przykładem takiej sytuacji może być dołączanie do systemu *diagramów sterujących*.

5.6.4 Projektowanie z użyciem diagramów K-GK

W celu ułatwienia planowania procesów przetwarzania *kształtów graficznych* prowadzonych gramatyką K-GK opracowany został specjalny system diagramowania. Proponowane w rozwiązaniu diagramy prezentują informacje o zależnościach pomiędzy operacjami uruchomienia poszczególnych *reguł substytucji* w procesie przetwarzania *kształtów*. Nie prezentują one bezpośrednio poszczególnych iteracji procesu. Umożliwiają natomiast określanie poszczególnych ścieżek, którymi toczą się procesy generowania poszczególnych komponentów końcowego diagramu *kształtów* wykonywanych równolegle podczas trwania tych samych iteracji. Jednocześnie – każda ze ścieżek może trwać dowolną ilość iteracji, dostarczając *kształt* wymagany przy generowaniu kolejnego komponentu. Na diagramie możliwe jest definiowanie linii synchronizacji pomiędzy ścieżkami, prezentujących wspomniane zależności. Dla zwiększenia przejrzystości wprowadzono symbolikę umożliwiającą wyrażenie procesu wielokrotnego uruchomienia tej samej *reguły substytucji* lub sekwencji takich samych *reguł substytucji*. Będzie to miało istotne znaczenie w sytuacjach, gdy o kolejnym uruchomieniu reguły decyduje wynik zwracany przez *funkcję walidatowra* (czyli gdy uruchomienie jest uzależnione od geometrycznego rozlokowania *kształtów graficznych* na *diagramie kształtów*). W poszczególnych kolumnach diagramu prezentowane są operacje, które kolejno muszą być wykonane, aby wygenerowany został określony komponent *diagramu kształtów*. Diagram obrazuje przebieg procesu przetwarzania gramatyką K-GK prowadzonego na konkretnym *diagramie kształtów*. Jego zawartość zmieni się po wymianie początkowego *diagramu kształtów* w gramatyce. Określa on zatem całą gramatykę K-GK, a nie jedynie z jej zbiór *reguł substytucji*. W związku z powyższym diagram ten nazwano *diagramem K-GK*.

Modelowanie przy użyciu diagramu K-GK ułatwi projektowanie gramatyk przetwarzających równolegle wiele komponentów *diagramu kształtów*. W konsekwencji pomoże zredukować ilość iteracji, konieczną do zrealizowania postawionego zadania. Przejrzysta forma prezentacji równolegle realizowanych transformacji ułatwi także określanie właściwej kolejności *reguł substytucji* na liście na liście R gramatyki K-GK.

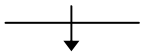
Diagram K-GK może zawierać następujące elementy:

- cztery symbole oznaczające aktywację *reguły substytucji* określonego typu. Są to symbole aktywacji reguły: redukcji (-=>), powielania (+=>), konwersji (:=>) i

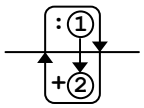
połączenia (*=>). Symbole określają także numery reguł na liście R *reguł substytucji* K-GK:

-① +② :③ *④

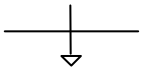
- symbol związku uzależniającego uruchomienie *reguł substytucji*:



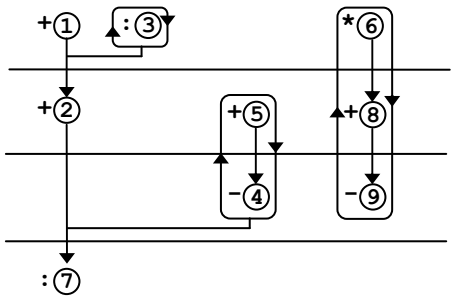
- symbol procesu wielokrotnego aktywowania sekwencji *reguł substytucji* (tu przedstawiony na przykładzie tylko dwóch reguł):



- symbol procesu generalizacji *kształtów* prowadzonego pomiędzy iteracjami (użycie symbolu wymaga zakończenia wszystkich ścieżek przetwarzania):



Na rysunku 5.16 znajduje się przykład diagramu K-GK prezentujący związki między operacjami aktywowania *reguł substytucji* oraz cykliczne uruchamianie sekwencji tych samych reguł (5+4 i 6+8+9).



Rysunek 5.16: Przykładowy diagram K-GK (źródło własne).

Diagramy K-GK otrzymują swój kształt dopiero po zbudowaniu pełnej gramatyki K-GK, czyli po ustaleniu treści początkowego *diagramu kształtów*. Umożliwiają określenie procesów przebiegających w ramach przetwarzania diagramu początkowego gramatyką K-GK. Procesy te na diagramie składają się z serii kolejno uruchamianych *reguł substytucji*. Wydzielona sekcja

pozioma (wiersz) diagramu nie powinna być interpretowana jako iteracja, lecz jako seria niezależnych od siebie operacji – możliwych do wykonania na diagramie bez wzajemnego oczekiwania na wyniki. Wydzielona kolumna diagramu to operacje od siebie uzależnione. Kolumny mogą być łączone i rozdzielane – skutkując wprowadzeniem uzależnienia operacji zgodnie z przebiegiem linii łączących.

5.7 Możliwości prowadzenia badań w dziedzinie automatycznego pozyskiwania kształtów graficznych do wykorzystania w przetwarzaniu K-GK

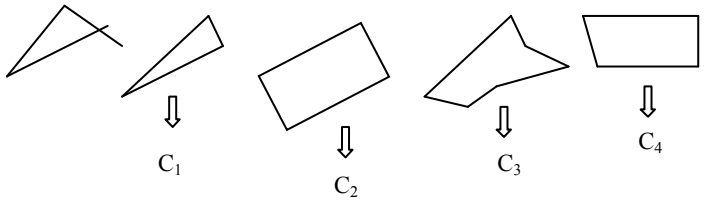
W niektórych sytuacjach teoretycznie możliwe jest tworzenie transformacji ułatwiających wygenerowanie reprezentacji graficzncyh *kształtów* na podstawie modelu rzeczywistego. Ich wyodrębnienie będzie zakładało istenienie dostarczonej z zewnątrz transformacji dopuszczalnej, tak jak w przypadku klasycznej *gramatyki kształtu* opisanej w punkcie 2.4.1. W ramach prac nad ostateczną formułą *reguły substytucji* stworzono kilka przykładowych transformacji, umożliwiających napisanie algorytmu przetwarzającego obiekt rzeczywisty zadany trójwymiarową siatką wielokątów w rozpoznawalne *kształty graficzne*. Istnieje wiele technik reprezentowania *kształtów* w trójwymiarowej przestrzeni projekcji, zwłaszcza gdy stanowią one bryłę. Obok reprezentacji siatkami wiekokątów można przykładowo zastosować woksele [Fol90], listy brył prostych łączonych operacjami boolowskimi [Fol90], ograniczać przestrzeń płaszczyznami podziału [Bis04], czy stosować drzewa ósemkowe albo BSP [Fol09]. Jednak z uwagi na popularność zastosowań danej techniki, a tym samym na powszechność występowania materiału graficznego oraz łatwość jego renderowania przy użyciu ogólnie dostępnych urządzeń i narzędzi zdecydowano się na prowadzenie badań wyłącznie nad interpretacją struktury siatek wielokątów. Dekompozycja takiego materiału graficznego ma nacelu:

- przeprowadzenie fizycznego podziału opisującej go trójwymiarowej siatki wielokątów,
- zidentyfikowanie wydzielonych siatek jako reprezentacje graficzne *klas kształtów* (poprzez porównanie siatek z opracowaną do tego celu metryką),
- ustalenie (na przykład poprzez obliczenie techniką bounding box [Lue03]) geometrycznych właściwości *kształtów graficznych*,

- zapisanie *kształtów* wraz z meta-danymi w *diagramie kształtów* K-GK.

Powstały *diagram kształtów* można wtedy traktować jako początkowy w gramatyce K-GK. Kilka tego typu rozwiazań autor opracował i zaimplementował w ramach prowadzenia doświadczeń [Tur09]. Wyniki nie były obiecujące i trzeba tu raczej wyciągnąć wniosek o braku możliwości opracowania rozwiazania na tyle uniwersalnego, aby warte było przyjęcia na listę tez dołączonych do niniejszej pracy. Ogrom szumów istniejących w tak zadanym materiale graficznym, pełna dowolność określenia *kształtu graficznego* (jaką udostępnia siatka wielokątów) to główne napotkane problemy. Zastosowanie takich metod możliwe jest jedynie wobec jednoznacznie zidentyfikowanego obiektu (reprezentowanego przez wspomnianą siatkę wielokątów). Ich znaczenie należy interpretować jedynie jako wytyczną to tworzenia narzędzi developerskich dla gramatyk, przy projektowaniu których projektant samodzielnie przeprowadzi podział siatki wielokątów i zakwalifikuje wydzielone komponenty jako kszałty.

Kształty początkowe gramatyki kształtu to symbole rozpoznawalne przez tą gramatykę (należące do jej alfabetu). Rozpoznawalność będzie podstawą do identyfikacji danego symbolu i zastosowania wobec niego określonej reguły (produkcji). Jak stwierdzono, nie jest możliwe pobranie z natury dowolnego *kształtu* i umieszczenie go w gramatyce. Teoretycznie możliwe jest uproszone prowadzenie analiz kształtów naturalnych - po znacznym zawężeniu przestrzeni projekcji. Jednym z pomysłów jest tutaj zawężenie wymiaru przestrzeni - na przykład do analizy *kształtów* w przestrzeni dwuwymiarowej. W przypadku poszukiwania nieskomplikowanych *kształtów* początkowych, których liczba ograniczona jest pulą od kilku do kilkunastu możliwości spodziewane są zadowalające wyniki. W takim wariancie rozwiązanie bazuje na kombinatorycznym dopasowywaniu sygnału wejściowego do wzorców, sprawdzajac zgodność jedynie kilku wyróżnionych atrybutów *kształtu*. Przykładowo – otrzymując *kształty* będące pojedyńczymi wielokątami. Nie posiadają one wielu atrybutów, gdyż są określane jedynie poprzez geometryczne usytuowanie swoich wierzchołków i naniesienie odcinków pomiędzy tymi wierzchołkami. Pokazano to na rysunku 5.17.



Rysunek 5.17: Identyfikacja *kształtu graficznego* na bazie sygnału uproszczonego: obraz 2D z łatwo rozpoznawalnymi obiektami (trójkąt, trapez, sześciokąt, prostokąt itp.) (źródło własne).

Warto zauważyć, iż *kształty* te po sklasyfikowaniu można łatwo umieścić w trójwymiarowej przestrzeni projekcji – dodając ujednolicone wartości z do wierzchołków siatki reprezentującej obiekt. Przetwarzanie *diagramu kształtów* nie będzie w żaden sposób ograniczone wykorzystaniem takich reprezentacji graficznych (jak zaznaczono wcześniej w Tabelach 5.1, 5.2), a wynikowy *diagram kształtów* będzie wizualizowany w trójwymiarowej przestrzeni projekcji.

Jeszcze innym podejściem może być paradoksalnie zignorowanie problemu braku determinizmu przy takiej transformacji. Transformacja może wtedy przykładowo jedynie pobieżnie analizować otrzymaną siatkę 3D poszukując w niej na przykład wierzchołków o znaczeniu wyróżnionym przez osobno zdefiniowane kryteria, ustalać tylko wymiary siatki (np. poprzez ustalenie maksimum odległości wierzchołków skrajnych) lub szukać punktów koncentracji dużej liczby wierzchołków [Dua02]. Te rozwiązania będą jednak również miały zastosowanie tylko w przypadku konkretnych zadań projektowych.

Tematyka związana z opracowywaniem rozwiązań opisanego powyżej problemu jest niezmiernie obszerna [Tur08] i niewątpliwie wykracza poza ramy rozważań w niniejszej pracy. Warto jednak zaznaczyć, iż proponowany w system może zostać zimplementowany tak, aby istniała możliwość wykorzystania takich metod jako alternatywne źródło *kształtów*. Dzięki postulatowi wprowadzenia poziomów abstrakcji w interpretowaniu materiału graficznego angażowanie takich rozwiązań w systemach generacyjnych będzie wymagało przeprojektowania tylko warstwy przetwarzajacej reprezentacje graficzne *kształtów*.

5.8 Ontologiczne podejście do problemu rozróżnialności klas kształtów

Zakres problemów związanych z wykazaniem jakie znaczenie w danej dziedzinie wiedzy ma konkretny, rozważany aktualnie obiekt został już wstępnie zarysowany w poprzednich rozdziałach. Każda dziedzina wiedzy, zawierająca definicje obiektów, które potencjalnie można przetwarzać wprowadza szereg specyficznych relacji, koniecznych do określenia powiązań między tymi obiektami. Większość takich obiektów nie jest definiowana formalnie, a ich znaczenie można oprzeć głównie na owych relacjach oraz opisowej wiedzy eksperta w danej dziedzinie. Wspomniano już także, iż pomocna przy uporządkowaniu odpowiedników znaczeniowych obiektów w tak chaotycznie wyrażonym układzie może być teoria ontologii. Jej

zastosowanie sprowadzi się do zaangażowania narzędzi, umożliwiających utrwalanie i przechowywanie identyfikatorów obiektów oraz relacji pomiędzy obiektami postrzeganymi przez eksperta w danej dziedzinie. Relacje wyrażą związki pomiędzy fizycznymi obiektami - a co za tym idzie – powiązania pomiędzy ich słownymi określeniami i przyszłymi *kształtami*. Teoria ontologii zakłada [Smi04], iż pomiędzy wieloma czynnikami wpływającymi na jednoznaczność przekazu dwa zasługują na szczególną uwagę. Są to kategoryzacja oraz hierarchizacja. Ponadto kryteria projektowania ontologii, jakie w 1993 zaproponował T. Gruber [Gru93] to jasność, spójność, rozszerzalność, minimalne zaangażowanie symboliczne i ontologiczne. Tworząc choćby projekt edytora *kształtów graficznych* umożliwiającego zakodowanie odpowiedników znaczeniowych tych *kształtów* można brać za wzór zestaw wymienionych przez Grubera wytycznych. Łatwo zauważyć, iż wymagania regulujące ontologiczne podejście do opisu klas obiektów wymuszają zdefiniowanie konkretnego formatu wiedzy. Jej struktura perfekcyjnie odpowiada zapotrzebowaniom, jakie pojawią się przy próbie dokonania formalnego określenia porządku w dziedzinie obiektów przekształcanych gramatyką K-GK. Opisowym odpowiednikiem domeny ontologii będzie tu po prostu dziedzina przetwarzanych *kształtów*. Dodatkowo podążanie ścieżką ontologii znaczeniowych umożliwi zaangażowanie w rozwiązaniach gotowych narzędzi wspomagających. To spostrzeżenie wywodzi się na przykład z przeglądu możliwości pakietów OntoSelect [Ont04]] lub Ontolingua [Ont08]. Rozwiazania te są potencjalnie możliwe do wykorzystania jako serwery ontologii dla *kształtów*. Tworzą rozproszone środowisko, umożliwiające budowanie i użytkowanie własnych ontologii. Możliwość wykorzystania takich narzędzi z pewnością zasługuje na dalsze rozważania.

6 Modele doświadczalne i oprogramowanie symulacyjne

W niniejszym rozdziale zostaną opisane rozważania prowadzące do sformułowania wytycznych umożliwiających projektowanie systemów przetwarzających z wykorzystaniem gramatyki K-GK. Ponadto omówiona w poprzednim rozdziale koncepcja procesu przetwarzania *kształtów graficznych* znajdzie swoją kontynuację w rozważaniach nad konkretnymi komponentami przykładowego systemu przetwarzania. Konieczne tu będzie opracowanie kolejnych rozwiązań umożliwiających wykorzystanie proponowanego procesu w zastosowaniach inżynieryjnych. Implementacja przedstawionych w rozdziale narzędzi i komponentów ma na celu wykazanie, iż można zbudować system przetwarzający złożone *kształty graficzne*, który będzie posiadał pożądane cechy funkcjonalne.

Jednym z przyjętych przed projektowaniem założeń było umożliwienie szerokiego spektrum zastosowań modelu K-GK w implementacjach. Wspomniano, iż właściwy efekt może zostać uzyskany poprzez wprowadzenie porządku interpretowania materiałów graficznych na trzech poziomach abstrakcji. Uniezależnienie warstwy przetwarzającej reprezentacje graficzne *kształtów* (zgodnych z definicją 5.1) od warstwy modyfikującej *diagram kształtów*, a następnie tej warstwy od najbardziej abstrakcyjnej warstwy zarządzającej *klasami kształtów* (definicja 5.2) powinno znacznie ułatwić projektowanie i implementację transformacji ramach poszczególnych warstw.

Dodatkowo - prowadzenie transformacji bazujących na *regułach substytucji* (definicja 5.7) będzie można akcelerować sprzętowo. Stanie się to przydatne podczas realizacji operacji geometrycznych prowadzonych przy przetwarzaniu *kształtów graficznych*. Korekta wierzchołków składowych bryły, czy inne operacje zmieniające jej postać są w zasięgu dzisiejszych graficznych platform sprzętowych. Co ważne – operacje te będą mogły być prowadzone w czasie rzeczywistym.

6.1 Architektura proponowanego systemu przetwarzającego kształty graficzne

Zróżnicowanie poziomów abstrakcji w interpretowaniu oraz przetwarzaniu zawartej w gramatykach K-GK wiedzy wpłynie bezpośrednio na technikę projektowania systemów

wykorzystujących takie gramatyki w zastosowaniach inżynieryjnych. Na wczesnych etapach prac badawczych (skutkujących ewoluowaniem obecnej koncepcji gramatyk) zostało opracowane kilkanaście komponentów weryfikujących zasadność stawianych założeń. Ustabilizowanie postaci ostatecznej spowodowało rewizję funkcjonalności poszczególnych komponentów, jej rozwarstwienie i utworzenie interfejsów pomiędzy komponentami.

Architektura warstwowa (począwszy od całkowicie abstrakcyjnej interpretacji *klas kształtów*, a skończywszy na renderowaniu powstałych *kształtów graficznych* w urządzeniach) umożliwi tworzenie systemów o czytelnej strukturze oraz całkowicie niezależną implementację ich komponentów. Reużywalność takich komponentów będzie ich naturalną cechą. Głównie za sprawą faktu, iż funkcjonalność abstrakcyjnych warstw „wyższych" może być różnorodnie wykorzystywana (Rysunek 6.1) – w zależności od treści komponentu realizującego transformacje geometryczne w już konkretnej przestrzeni, a potem kolejnego – prowadzącego renderowanie wyników w fizycznym urządzeniu.

Przetwarzanie wiedzy o *klasach kształtów*
Przetwarzanie reguł substytucji (z wyłączeniem przetwarzania treści *funkcji walidatora* i *funkcji konwertera*)
Przetwarzanie *kształtów* w ramach *funkcji walidatorów* i *funkcji konwerterów*
Reprezentacja graficzna *kształtów* i ewentualna dalsza obróbka klasyczną gramatyką kształtu
Renderowania *kształtów* w urządzeniu

Rysunek 6.1: Warstwowa architektura graficznego systemu generacyjnego, wykorzystującego model K-GK (źródło własne).

6.1.1 Komponent realizujący przetwarzanie klas kształtów

Komponent przetwarzający *klasy kształtów* będzie posiadał dostęp do wiedzy o klasyfikacji jakościowej obrabiwany gramatyką elementów. Jego funkcjami są:

- dostarczanie informacji o nadklasach *kształtów* oraz *klasach kształtu* wyprowadzonych z tej samej nadklasy,

- utrzymywanie komunikacji z innymi równoważnymi funkcjonalnie komponentami, lecz przetwarzającymi inne bazy wiedzy,
- poszukiwanie odpowiedników znaczeniowych (ontologicznych) dla *klas kształtów* występujących w innych bazach wiedzy.

Jak widać, komponent przetwarza pewną wiedzę o *kształtach graficznych*, jednak nie analizuje żadnych konkretnych obiektów umieszczonych na *diagramie kształtów*, a tym bardziej ich geometrycznego rozlokowania w przestrzeni projekcji (ujętego definicją 5.4). Nie odnosi się również w żaden sposób do reprezentacji graficznej *kształtów*. Abstrahuje od wszelkich aspektów związanych z renderowaniem, interpretując wyłącznie klasyfikację jakościową przetwarzanego materiału. Daje tym samym możliwość wykorzystania swojej funkcjonalności w jak najszerszym spektrum zastosowań.

6.1.2 Komponent realizujący przetwarzanie diagramu kształtów regułami substytucji

Głównym komponentem systemu będzie bez wątpienia „silnik" przetwarzający *reguły substytucji*. Operując na *diagramie kształtów* zajmie się on przeszukiwaniem kolejki *Reguł substytucji* i wprowadzaniem korekt do tego diagramu na podstawie treści reguł. Ponadto będzie źródłem decyzji dotyczących prowadzenia operacji uogólniania *kształtów*. Zarówno kwestia przeszukiwania kolejki *reguł substytucji* jak i prowadzenia uogólniania musi być dokładnie rozważona.

Uogólnianie kształtu

Generalną przesłanką do uogólnienia *klasy kształtu*, czyli wyszukania jego nadklasy jest brak możliwości dalszego rozwijania *kształtu* na podstawie dostępnej o nim wiedzy. Innymi słowy – brak reguły zawierającej w LHS kombinację umożliwiającą przeprowadzenie *kształtu* w następny, a w konsekwencji – w *kształt terminalny*. Wówczas warta wypróbowania jest adaptacja innej wiedzy o podobnym znaczeniowo obiekcie do przetworzenia bieżącego *kształtu*. Praktycznym przykładem zobrazowania takiej sytuacji może być konieczność zbudowania modułu „hamulca w samochodzie osobowym" (jako uzupełnienie „koła samochodu") z zapożyczeniem rozwiązania z wiedzy służącej do budowy takiego układu w „pojeździe", a następnie (tu następuje wyspecjalizowanie) w „ciężarówce".

Emisja decyzji o uogólnieniu wyczerpuje możliwości deterministycznego prowadzenia całego procesu. Uogólniać obiekt, a potem specjalizować go można na wiele sposobów – zależnie od wiedzy o odpowiednikach znaczeniowych, posiadanej przez komponent przekształcający *klasy kształtów*. Dlatego procedura uogólniania powinna być kontrolowana w sposób ścisły. Od jej przebiegu zależeć będzie wygląd końcowy d*iagramu kształtów*. W implementacjach zakłada się wprowadzenie jednego z dwóch rozwiązań:

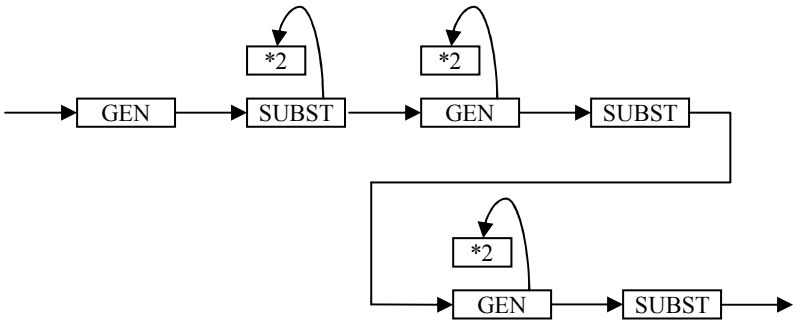
- wprowadzenie opisywanych już w podrozdziale 5.4.10 *diagramów sterujących*, określających sekwencję sterującą operacjami uogólniania i zwykłej substytucji dla całego procesu,
- wyprowadzenie procesu decyzyjnego do interfejsu użytkownika – umożliwiając w połączeniu z interfejsem graficznym przetwarzającym graficzne reprezentacje *kształtów* podejmowanie decyzji w trybie interaktywnym.

Pierwsze rozwiązanie ma zastosowania w przypadku użytkowania spójnej bazy wiedzy, zaprojektowanej pod kątem rozwijania obiektów podobnej klasy (inna sprawa to rodzaj, ilość i lokalizacja fizycznych komponentów takich obiektów). Rozwiązanie to było regularnie stosowane podczas badania systemów testowych. Sekwencja takich czynności jest wykonywana przez komponent zarządzający przetwarzaniem *reguł substytucji*. Język ją opisujący stał się rozszerzeniem języka CSGL, więc jego instrukcje mogły zostać zapisane w postaci plikowej nad XML (Rysunek 6.3).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<csgl:csg xmlns="http://home.agh.edu.pl/mitu/csgl-namespace" dis-
tance="here">
    <executors>
        <executor generalize=1>komentarz</executor>
        <executor loop=2></executor>
        <executor generalize=1 loop=2></executor>
        <executor></executor>
        <executor back=2></executor>
    </executors>
</csgl:csg>
```

Rysunek 6.3: Przykład zapisu *diagramu sterującego* w rozszerzeniu języka CSGL (źródło własne).

W ramach sekwencji tych czynności dopuszczane są także skoki (parametryzowane liczbą rozkazów o jaką należy wykonać skok) oraz pętle wymuszające powtórzenie konkretnego rozkazu. Powyższy przykład (z rysunku 6.3) wymusi przebieg procesu przetwarzania zobrazowany na rysunku 6.4.



Rysunek 6.4: Przebieg operacji dla przykładu z rysunku 6.3 (źródło własne).

Drugie rozwiązanie (bazujące na interaktywnym pobieraniu decyzji od użytkownika) umożliwia ścisłą kontrolę nad procesem przetwarzania, pożądaną przykładowo w narzędziach wspomagających projektowanie. Użytkownik końcowy może samodzielnie zarządzać kolekcją baz wiedzy, uporządkowanych w ontologicznie skomponowaną całość. Może wówczas podjąć decyzję o uogólnieniu rozwiązań (w przypadku braku treści w lokalnej bazie wiedzy) – z jednoczesnym wskazaniem konkretnej wiedzy powiązanej, z której brakujący fragment powinien zostać zaczerpnięty. Odwołując się tu znów do prozaicznego przykładu „koła do samochodu" – uporządkowana wiedza będzie skupiała powiązane za sobą rozwiązania umożliwiające skonstruowanie „koła" w przypadkach gdy jest ono komponentem najróżniejszych urządzeń czy konstrukcji. Gdy wiedza stosowana do tworzenia aktualnie rozwijanego przez użytkownika obiektu jest niewystarczająca (brak opisu detali) – decyduje on wykorzystaniu wiedzy zastępczej - wybierając z listy rozwiązanie alternatywne. Lista utworzona zostanie na podstawie relacji pomiędzy *klasami kształtów*, a wybór spowoduje zamianę klasy przetwarzanego obiektu na wytypowany przez użytkownika nadzorującego.

Poszukiwanie reguł

Diagram sterujący określa kolejny krok postępowania przy rozpoczynaniu iteracji w procesie przetwarzania *kształtów*. Głównym jego przeznaczeniem jest wytyczenie momentów, w których należy generalizować klasy przetwarzanych *kształtów* w celu poszukiwania rozwiązań spoza bieżącego zbioru *reguł substytucji* (wykorzystanie rozwiązania

uzupełniającego brak w aktualnie wykorzystywanej wiedzy). Wspomniana iteracja będzie polegała na jednokrotnym zaangażowaniu (niekoniecznie przekształceniu) wszystkich *kształtów graficznych* w *diagramie kształtów* bądź wykazaniu braku możliwości ich zaangażowania. Zgodnie z [Sti80] dopasowywane będą *kształty* do reguł, nie reguły do *kształtów*. Zatem wymagany jest algorytm, który dla każdej *reguły substytucji* przeprowadzi proces dopasowywania *kształtów*. Czynnikiem determinującym dopasowanie, zgodnie z założeniami powziętymi w poprzednim rozdziale, będzie zawartość LHS reguły oraz wartość zwracana przez *funkcję walidatora*. *Funkcja walidatora* operuje na niższym poziomie abstrakcji, kontrolując dopasowanie geometryczne *kształtów*. Z punktu widzenia obecnie rozważanego modułu jest jedynie jednoznacznie identyfikowanym operatorem, korzystającym z nie interpretowanych tu meta-danych *kształtu graficznego.* Operator ten pobiera jako operandy zadeklarowane *kształty graficzne* z LHS *reguły substytucji* i zwraca wartość logiczną. Dopasowanie będzie zatem wymagało spełnienia dwóch warunków:

- odnalezienia w *diagramie kształtów* wszystkich wymienionych w LHS k*ształtów graficznych*,
- poprawnej walidacji odnalezionych *kształtów* przez funkcję *walidatora*.

Poszukiwanie wymaga utworzenia wszystkich wariancji *kształtów graficznych* dla każdej *klasy kształtu* występujacej w LHS reguły. Następnie podstawienia ich do *reguły substytucji* celem sprawdzenia *funkcją walidatora*. Przykładowo dopasowanie reguły:

$$C_1C_1C_2 \ :=> \ C_1C_1C_3,\ w(C_1,C_1,C_2),\ k(C_1C_1C_2,\ C_1C_1C_3)$$

do *diagramu kształtów*:

$$a1{:}C_1,\ a2{:}C_1,\ a3{:}C_1,\ a4{:}C_2,\ a5{:}C_2,\ a6{:}C_3$$

daje następującą pulę możliwości, którą należy sprawdzić *funkcją walidatora*:

```
(a1, a2, a4), (a1, a3, a4), (a2, a1, a4),
(a2, a3, a4), (a3, a1, a4), (a3, a2, a4),
(a1, a2, a5), (a1, a3, a5), (a2, a1, a5),
(a2, a3, a5), (a3, a1, a5), (a3, a2, a5)
```

W przypadku poprawnej walidacji przez *funkcję walidatora reguła substytucji* jest uruchamiana i przeprowadzana jest stosowna operacja zamiany *kształtów graficznych*. Potem następuje uruchomienie *funkcji konwertera*. Żaden uczestniczący w konwersji *kształt* (także ten nie zmodyfikowany) nie może już brać udziału w dalszym procesie dopasowywania. Dotyczy to

także pozostałych *reguł substytucji* w danej iteracji. Pula możliwości wykorzystania reguł zawęża się im dalszą pozycję w kolejce reguła zajmuje. Tym samym zyskujemy możliwość nadawania priorytetów *regułom substytucji*. Uruchomienie *funkcji konwertera* dokonuje korekt w m-elementowym zbiorze meta-danych (P) *kształtu graficznego*. Procedura ta prowadzona jest przez komponent, będący przedmiotem rozważań następnego podrozdziału.

6.1.3 Komponent realizujący przetwarzanie kształtów w ramach funkcji walidatorów i konwerterów

Kolejny komponent konieczny do implementacji systemu funkcjonuje już w konkretnej i formalnie zdefiniowanej przestrzeni n-wymiarowej. Na wyższych poziomach abstrakcji geometryczne usytuowanie przetwarzanych *kształtów* w przestrzeni projekcji było kompletnie nieistotne. Dlatego nawet wymiar samej przestrzeni nie musiał być znany. Interpretowano tam jedynie wiedzę o *klasach kształtów* (najwyższy poziom) oraz *diagram kształtów* będący uporządkowaną listą *kształtów graficznych* danych klas (poziom bezpośrednio niższy). W tym drugim przypadku rozważano istnienie potencjalnej możliwości konwersji *kształtów graficznych* w inne oraz modyfikacji ich meta-danych dokonywanej przez *funkcję konwertera*, od której treści abstrahowano. Podobnie abstrahowano od treści *funkcji walidatora*, czyniącej z „potencjalnej możliwości dokonania konwersji" możliwość faktyczną, albo wykazującej brak takiej możliwości.

Obecnie funkcje te są konkretnymi metodami klas implementowanymi w języku programowania, w którym kodowany będzie komponent. Otrzymują w parametrach zadane meta-danymi informacje o usytuowaniu *kształtu graficznego* w przestrzeni projekcji i przetwarzają te meta-dane. Wywoływaniem tych metod steruje inny komponent (przetwarzający *reguły substytucji)*, dlatego bieżący komponent najlepiej jest w implementacjach przygotować pod postacią biblioteki klas i metod.

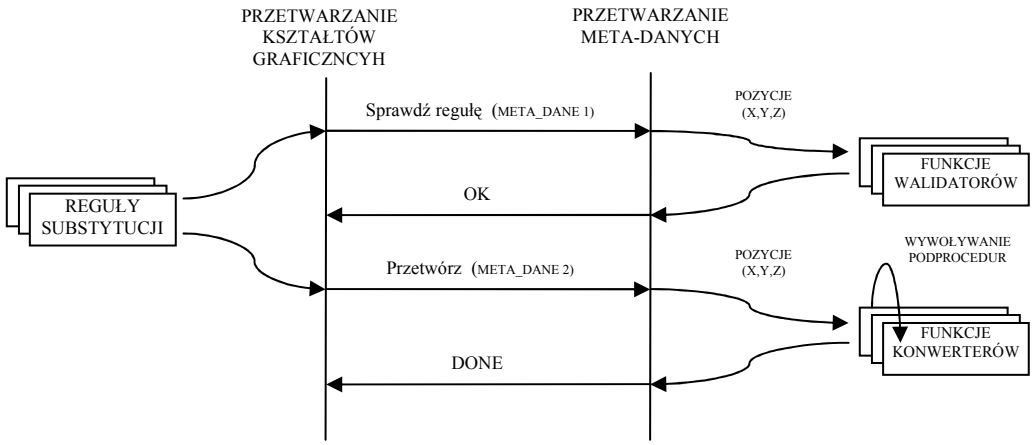
Komponent ten musi utrzymywać rejestr *funkcji konwertera* i *funkcji walidatora*. Funkcje te będą znakowane unikatowymi identyfikatorami (wystepując na przykład pod postacią tak zwanych eksportów bibliotecznych). Funkcje nie interpretują *klas kształtów* - nie mając do nich dostępu. Zgodnie z definicjami 5.8, 5.9, 5.10 i 5.11 poszczególnych *reguł substytucji* zawartych w ogólnej koncepcji K-GK *funkcje walidatora* i *funkcje konwertera* operują jedynie na meta-

dancych wykonując na nich operacje geometryczne. Takie ograniczenie postawione zostało celowo i jest konsekwencją kilku istotnych kwestii:

- odseparowanie funkcjonalności komponentu skutkujące łatwiejszą implementacją,
- wyjście naprzeciw obserwowanej tendencji coraz silniejszej akceleracji sprzętowej operacji geometrycznych. Oznacza to, iż funkcjonalność komponentu będzie można kodować za pomocą tzw. vertex shaderów [Nvi06],
- uzyskanie możliwości użycia tej samej *funkcji konwertera* lub *funkcji walidatora* w odniesieniu do *kształtów graficzny*ch różnych *klas kształtów* poprzez odwołanie z różnych *reguł substytucji*.

W komentarzu do ostatniego punktu listy warto zauważyć, że po ukonkretnieniu przestrzeni projekcji meta-dane mają w przypadkach wszystkich *kształtów graficznych* homogeniczną strukturę. Tym samym także *kształty graficzne*, będące tutaj już obiektami opisanymi w języku programowania, pozyskają identyczną strukturę (stając się obiektami tego samego typu). Typ ten będzie jednocześnie spójnym typem argumentu dla funkcji *walidatorów* i *konwerterów* zawartych w bibliotece. Wywoływanie tych samych funkcji w kontekście walidacji lub konwersji *kształtów graficznych* naprzemiennie różnych *klas kształtów* będzie możliwe i poprawne. Zatem tą samą funkcją (gdy baza *reguł substytucji* dopuści taką możliwość) będzie można analizować lokalizację różnych znaczeniowo obiektów.

Jak ustalono, współpraca komponentu tej warstwy w komponentem sterującym (w wyższej warstwie abstrakcji) będzie dotyczyła walidacji oraz modyfikacji meta-danych towarzyszących *kształtom graficznym*. Poglądowy diagram blokowy wyrażający wzajemną interakcję i przepływ danych przedstawiono na rysunku 6.5.



Rysunek 6.5: Przepływ danych i interakcja pomiędzy komponentami uruchomieniu *reguły substytucji* (źródło własne).

Wiedza charakteryzująca *kształty graficzne* i przetwarzana przez obecnie rozważany komponent oraz wcześniej opisany komponent z wyższej warstwy abstrakcji stanowić będzie materiał wystarczający do wyrażenia graficznej reprezentacji *diagramu kształtów.* Funkcjonalność kolejnego modułu umożliwi renderowanie tak przygotowanego *diagramu*.

6.1.4 Moduł zarządzający reprezentacją graficzną elementów diagramu kształtów

Reprezentacja *kształtu graficznego* może być elementem dość dowolnie określanym. Wyjściowo dopuszcza się reprezentowanie jedynie *kształtów graficznych* nie abstrakcyjnych *klas kształtów*, czyli właśnie klas posiadających reprezentację graficzną. Stwarza to pewne problemy przy próbach wizualizacji form przejściowych *diagramu kształtów*, powstałych po kolejnych iteracjach procesu przetwarzania gramatykami K-GK. Brak jest wówczas reprezentacji graficznych *kształtów*, które mogą pojawić się na tym diagramie i powinny być wizualizowane. Problem można jednak łatwo ominąć wprowadzając reprezentacje zastępcze. Taki komponent *diagramu kształtów* jest tak czy inaczej jedynie tymczasowy i nie będzie występował w diagramie wynikowym. Wynikowy *diagram kształtów* może zawierać *kształty graficzne* podstawiane jako początkowe (*P*) do klasycznej gramatyki kształtu, zawartej w *klasie* tego *kształtu*. Proces ich rozwijania będzie oparty o dalsze wyprowadzenie w konkretnej klasycznej gramatyce kształtu. Będzie on przebiegał w oparciu o odrębnie zaimplementowane i ogólnie znane procedury ([Sti78]). Przetworzenie klasyczną gramatyką kształtu początkowego *P* może jednak pozostawiać ten *kształt* bez zmian - powodując natychmiastowe przeprowadzanie go w *kształt* (symbol) terminalny. Wówczas dalsze transformacje w ramach

reprezentacji graficznej *kształtu* nie nastąpią. W przypadku powzięcia takiego założenia w zakresie globalnym możliwe będzie reprezentowanie *kształtów graficznych* przy pomocy słownika reprezentacji graficznych, zawierającego na przykład gotowe zadane siatkami wielokątów modele trójwymiarowe lub opisane w dowolnej konwencji figury geometrycznie.

Implementacja modułu wizualizacji sprowadzi się do wyświetlania przy pomocy biblioteki graficznej reprezentacji graficznych listy *kształtów graficznych* w zadanym tą reprezentacją formacie. Meta-dane opisujące *kształty graficzne* z *diagramu kształtów* posłużą do ustalenia wartości pozycji, rotacji czy innych parametrów precyzujących lokalizację rysowanych *kształtów graficznych* w przestrzeni projekcji. Reprezentacja graficzna (np. siatka wielokątów lub jednoznacznie określony prymityw) będzie materiałem wykorzystywanym przez faktyczną procedurę renderującą.

6.1.5 Moduł realizujący renderowanie reprezentacji graficznej diagramu kształtów

Warstwowy podział funkcjonalności systemu przetwarzania *kształtów graficznych* pomiędzy poszczególne jego komponenty z pewnością znacznie ułatwi implementację gotowych rozwiązań inżynieryjnych renderujących *diagram kształtów*. Łatwość ta szczególnie uwydatnia się w przypadku analizy wymagań dla modułu renderującego. Renderowanie nie jest obowiązkowe, wiec w niektórych implementacjach moduł ten może być w ogóle nieobecny. Gdy wizualizacja jest pożądana – powinien on angażować dowolną bibliotekę graficzną umożliwiającą projekcję grafiki 2D lub 3D. Sama projekcja sterowana będzie zapewne poprzez dodatkowe interfejsy użytkownika, których zaprojektowanie jest sprawą zupełnie otwartą. Przedmiotem renderowania będą reprezentacje graficzne *kształtów* wyrażone w notacji przetwarzanej przez poprzedni komponent Moduł musi zatem poprawnie interpretować tą notację. Renderowanie będzie polegało na prowadzeniu dekompozycji reprezentacji graficznej na elementy możliwe do narysowania funkcjami biblioteki graficznej. W przypadku przykładowo wyrażenia *reprezentacji graficznej kształtu graficznego* poprzez siatkę wielokątów w trójwymiarowej przestrzeni euklidesowej moduł będzie:

- realizował proces dekompozycji trójwymiarowej siatki wielokątów na poszczególne wielokąty,
- realizował proces dekompozycji wielokątów na wierzchołki i koordynaty tekstur,

- iteracyjnie wywoływał funkcje biblioteki graficznej w celu rysowania poszczególnych produktów przeprowadzonej dekompozycji.

Rysowanie będzie dodatkowo prowadzone w lokalizacji globalnej przestrzeni euklidesowej wymuszonej przez komponent opisany w poprzednim podrozdziale i interpretujący zawartość meta-danych *kształtów graficznych* na *diagramie kształtów*. Stąd przykładowo zakodowana w tych meta-danych translacja *kształtu* na *diagramie kształtów* faktycznie spowoduje narysowanie *kształtu graficznego* we właściwym miejscu wirtualnej przestrzeni projekcji.

6.1.6 Ocena założonej architektury systemu

Komponenty opisane w poprzednich podrozdziałach zostały poukładane zgodnie z porządkiem warstw abstrakcji interpretowania elementów logicznych przetwarzanych w modelu K-GK. Spowoduje to utrzymanie modelu warstwowego także w architekturze samych komponentów przyszłych systemów. Natura danych wymienianych pomiędzy komponentami gwarantuje ich silne uniezależnienie, skutkując możliwością łatwej rozbudowy komponentów o implementacje dalszej funkcjonalności (rozszerzalność). Przepływy danych pomiędzy interfejsami komponentów są także łatwe do rozbudowania.

6.1.7 Interfejs wspierajacy wymianę informacji o ontologicznej reprezentacji wiedzy

W rozdziale piątym ustalono, iż przybliżenie teorii ontologii w wariancie „ontologii dziedzinowych" do rozważań prowadzonych w niniejszej pracy jest możliwe. Zaadaptowanie któregoś z już istniejących rozwiązań przetwarzających wiedzę uporządkowaną ontologicznie otworzy jeszcze jedną drogę do kontynuowania procesu transformacji *kształtów graficznych.* Droga ta będzie przydatna w sytuacji, gdy za pomocą bierzącej gramatyki K-GK nie będzie można wygenerować postaci wynikowej *diagramu kształtów* pomimo zastosowania mechanizmu generalizowania klas. Dzięki opisywanemu zabiegowi możliwe staje się określenie znaczenia wiedzy definiowanej przez eksperta i wyrażonej w postaci opisanych *kształtów*. Biblioteki wiedzy uporządkowanej ontologicznie, takie jak OntoSelect [Ont05] czy Protege [Ont10] utrzymują informacje o pojęciach (klasach obiektów), samych ontologiach (identyfikowanych najczęściej wartościami liczbowymi), oraz tzw. etykietach (*Labels*)

będących dodatkowymi określeniami klas. Podanie w zapytaniu owej etykiety umożliwia otrzymanie listy ontologii w jakich etykietowane pojęcia występują oraz list języków naturalnych (przykładowo: angielski, francuski etc.) w jakich te ontologie są zdefiniowane. Niektóre repozytoria ontologii udostępniane są publicznie i działają w oparciu o protokół *http*. Poza interaktywną komunikacją nad *http* możliwa jest także obróbka materiału otrzymanego w którymś z języków opisu ontologii, na przykład języku OWL (*Ontology Web Language*) [Owl09]. Tryb interaktywnej wymiany zapytań *http* jest jednak wystarczajacy dla zaspokojenia potrzeby uzyskania informacji o ewentualnych odpowiednikach *klas kształtu*.

Przykładowy interfejs przygotowano w oparciu o repozytorium OntoSelect. Ponieważ *klasa kształtu* jest identyfikowana wartością łańcuchową, możliwe jest odszukanie odpowiedników znaczeniowych danej nazwy klasy w repozytoriach ontologii. W tym celu składane jest zapytanie *http* do OntoSelect. Zakłada się przy tym, że znana jest jedna etykieta *klasy kształtu* wyrażona w konkretnym języku (np. angielski). Staje się ona ontologiczną etykietą (*label*) klasy w repozytorium i jest podawana do zaptania. W rezultacie uzyskiwany jest zbiór identyfikatorów i nazw ontologii, w których występuje ontologiczna klasa z nią powiązana (Rysunek 6.6).

Rysunek 6.6: Materiał pozyskiwany z repozytorium ontologii - podawany później do parsera HTML (interfejs WWW)[Ont04].

Pozyskanie ontologicznej wiedzy o *klasie kształtu* pozwoliło na kontynuację przetwarzania *diagramu kształów*. Jest to naturalnie możliwe, gdy w repozytorium ontologii odszukano kolejne etykiety (czyli łańcuchy tekstowe) określające poszukiwaną klasę w innych językach naturalnych. To pozwala na przeszukanie lokalnych zasobów CSGL (czyli zapisanych w języku zdefiniowanym w rozdziale 5.6.2) pod kątem występowania w nich *klasy kształtu* nazwanej analogicznie jak któraś z ontologicznych etykiet. Gdy taka klasa zostanie znaleziona – wiedza zawierajaca tą klasę zostanie załadowana w miejsce obecnie wykorzytywanej i ten sam *diagram kształtów* będzie rozwijany w kolejnej iteracji nową gramatyką K-GK.

Podsumowując – stworzony interfejs stanowi przykład zintegrowania rozwijanych narzędzi z zewnętrzną usługą - w celu jednoczesnego wykorzystania zróżnicowanej dziedzinowo wiedzy przygotowanej jako treść odosobnionych gramatyk K-GK. Implementacja

interfejsu sprowadza się do wykorzystania API klienta *http* przy formułowaniu zapytań *http*. Następnie – do wprowadzenia modyfikacji sterującej procesem aktualizacji wiedzy wykorzystywanej do tworzenia gramatyki K-GK, realizowanym w konsekwencji usyskania odpowiednika ontologicznego wybranej *klasy kształtu*.

6.2 Implemenatacja przykładowego systemu przetwarzającego kształty graficzne

W niniejszym podrozdziale zostaną przedstawione rozwiązania szczegółowe, których opracowanie jest konieczne do zaprezentowania przykładu implementacji doświadczalnego modułu przetwarzającego *diagramy kształtów* z wykorzystaniem gramatyk K-GK. Następnie zaprezentowany zostanie projekt przykładowego systemu.

Poszczególne rozwiązania techniczne gwarantujące implementację komponentów opisanych w podrozdziale 6.1 będą przedmiotem obecnych rozważań. Dotyczyć będą sposobów zakodowania rozproszonej wiedzy, tworzenia na jej podstawie struktur przetwarzanych przez komponenty przykładowego systemu. Poruszą także kwestie technicznych aspektów integracji tych komponentów, implementacji w komponentach algorytmów przetwarzających *diagram kształtów* na podstawie wiedzy oraz technik renderowania wyników czyli przetwarzania reprezentacji graficznych w *przestrzeni projekcji* zgodnie z treścią *diagramu kształtów*.

6.2.1 Przechowywanie wiedzy - moduł ładujący dane CSGLCodec

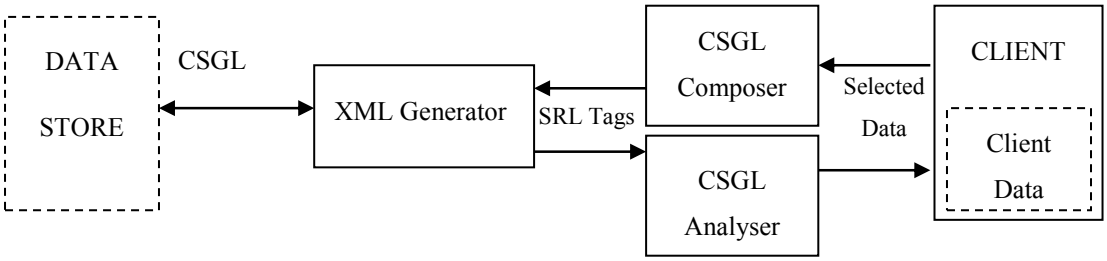
Użycie zdefiniowanego w poprzednich rozdziałach języka opisu *kształtów* i relacji pomiędzy *kształtami*, określonego mianem CSGL (*Cascade Shape Grammar Language*) wymaga zaprojektowania i wytworzenia narzędzia kodującego i dekodującego wiedzę o *kształtach* do postaci wyrażonej tym językiem i w kierunku przeciwnym.

Powstały w konsekwencji moduł bazuje na języku XML. Wykorzystuje gotowe komponenty dostarczające funkcje strumieniowania XML. Jego zadaniem jest:

- przetworzenie otrzymanej w serii instancji klas języka programowania wiedzy do postaci strumienia CSGL (zapis),

- zdekodowanie opisującego wiedzę strumienia CSGL i utworzenie na jego podstawie serii instancji klas języka programowania, opisujących swoją treścią zakodowaną uprzednio wiedzę (odczyt).

Powyższe funkcje realizują zatem procesy analogiczne do procesu marshallingu XML (serializacji obiektów do XML) i demarshallingu XML - lecz realizowane nad CSGL. Zastosowany zatem zostanie powszechnie dziś wykorzystywany mechanizmu serializacji obiektów [Llc10]. Funkcjonalność wspierająca serializację jest najczęściej dostarczana poprzez umieszczenie w klasie metody serializującej treść instancji tej klasy lub implementację interfejsu serializującego. W przypadku modułu będą to metody zapewniające import i eksport serii komponentów opisanych językiem CSGL. Marshalling prowadzony jest z użyciem gotowego parsera XML (na rysunku oznaczonego przez XML generator) i wykorzystanego w implementacji modułu ładującego (rysunek 6.7). Jest to otwarty produkt o nazwie TinyXML [Tix10].



Rysunek 6.7: Integracja modułu *CSGLCodec* z innymi komponentami systemu (źródło własne).

6.2.2 Projekt systemu przetwarzającego kształty

Niniejszy podrozdział zawiera propozycję architektury modułu przetwarzającego *diagramy kształtów*.

W projekcie systemu wspomagającego przetwarzanie z użyciem gramatyk K-GK zakłada się architekturę komponentową. Zdefiniowanie komponentów komunikujących się za pośrednictwem interfejsów umożliwi łatwą rozszerzalność systemu. Rozszerzalność ta ma znaczenie przede wszystkim przy kolejnych próbach implementacji algorytmów przetwarzających struktury uzależnione od różnoorodnych *przestrzeni projekcji* (*projective space)*[Har94] i *reprezentacji graficznej kształtów*.

W tworzonym systemie można wyróżnić następujące komponenty (reprezentowane przez klasy w języku programowania, w którym zakodowany jest system):

- `TransformationEngine` – moduł główny zarządzający logiką interpretowania *reguł substytucji*. Jest uniezależniony od dziedziny, z której wywodzi się przetwarzana informacja (nie interpretuje znaczenia *klas kształtów*, jedynie ustanawia przypisania *kształtów graficznych* do ich klas). Przetwarza `TransformationPool`, ujmując *kształty graficzne* abstrakcyjnie, czyli bez odniesienia do lokalizacji w *przestrzeni projekcji*.

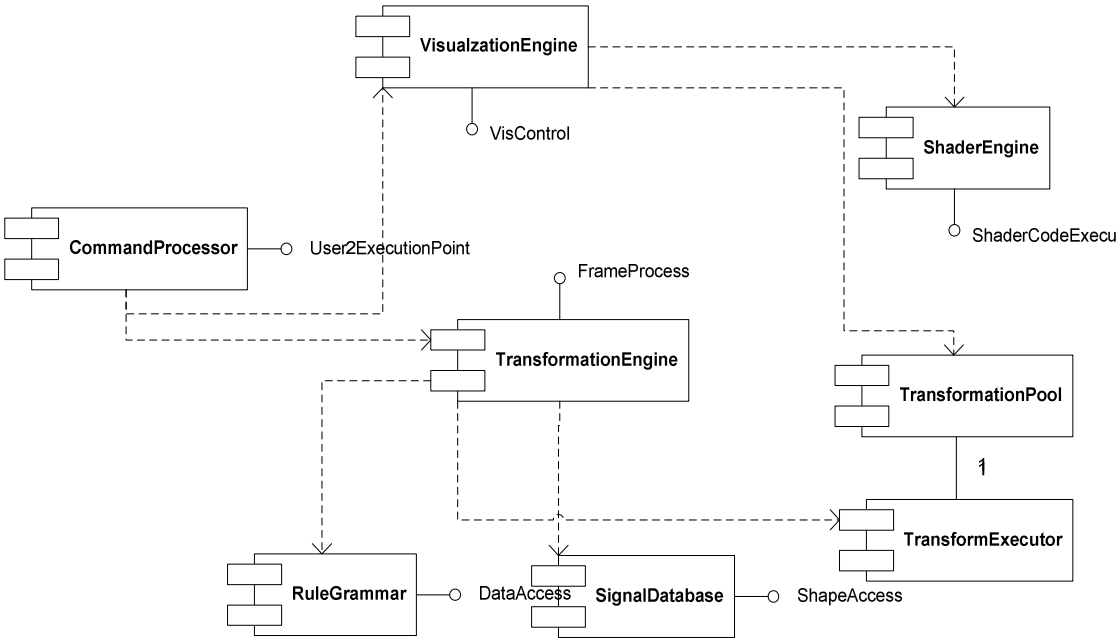
- `TransformationPool` – magazyn danych podlegających transformacji (czyli *diagramu kształtów*). W `TransformationPool` *kształty graficzne* posiadają przypisania do klasy i mogą je zmieniać na skutek stosowania wobec nich *reguł substytucji*. Dodatkowo meta-dane *kształtu* (na niższym poziomie abstrakcji) są interpretowane i modyfikowane zgodnie z treścią *funkcji konwerterów* i *funkcji walidatorów* wskazanych przez odpalaną *regułę substytucji*.

- `TransformExecutor` – komponent, który utrzymuje i interpretuje *diagram sterujący*, wykorzystywany w procesie uruchamiania iteracji lug generalizowania *klas kształtów*. Uzależnia od siebie `TransformationEngine`, będąc arbitrem procesów w nim zachodzących.

- `SignalDatabase` – baza wiedzy o *klasach kształtów* i ich naturze. Klasa poza swoim identyfikatorem oraz relacjami lokalizującymi ją na modelu *klas kształtów* danej gramatyki posiada jeszcze dodatkowe dane opisujące. Będą one stanowiły podstawę do poszukiwania odpowiedników znaczeniowych klas w ewentualnych innych bazach wiedzy. Informacje te zadane są dowolnie długim łańcuchem znaków, nazywanym podczas implementacji „sygnałem opisującym". Moduł zewnętrzny względem obecnego systemu, poszukujący odpowiednika znaczeniowego *klasy kształtu*, będzie te dane interpretował.

- `VisualizationEngine` – rozszerzenie wizualizujące aktualną zawartość TransformationPool lub jej zleconą część. Ten komponent może być pusty, jeśli nie przewiduje się natychmiastowej wizualizacji wyników.

- `ShaderEngine` – także opcjonalne rozszerzenie, będące modułem akcelerującym sprzętowo proces renderowania. Opisane zostanie w rozdziale 6.3.



Rysunek 6.8: Diagram komponentów graficznego systemu generacyjnego opartego o przetwarzanie gramatyk K-GK wraz z interfejsami (źródło własne).

System przetwarzający *diagram kształtów* został zaprojektowany tak, aby umożliwić jego swobodne rozszerzanie (Rysunek 6.8). Przy odpowiednim uporządkowaniu modelu *klas kszałtów* sam *diagram kształtów* (`TransformationPool`) może być także obiektem współdzielonym przez wiele homogenicznych strukturalnie komponentów przetwarzających. Skalowalność w tym rozumieniu zapewniona jest przez możliwość dodania współbieżnego toku przetwarzania różnych obiektów i różnymi zbiorami *reguł substytucji*. Przykłady takiego uporządkowania będą omawiane w rozdziale 7. Kwestia rozszerzalności systemu jest oczywista – każda warstwa jest strukturalnie niezależna od innych i może być wypełniona inną (zupełnie nową) funkcjonalnością.

Przy tak sformułowanej architekturze proponowany system będzie całkowicie uniezależniony od dziedziny wiedzy charakteryzującej faktycznie przetwarzany przez niego materiał (*kształty graficzne*). Aby przebudować system dostosowując go do innej dziedziny – należy wprowadzić modyfikacje przetwarzanej wiedzy, czyli:

- zastosować inny zbiór *reguł substytucji* (opisujących sposób generowania lub sterowania obiektem, jeśli ten się różni),
- dostarczyć bazę *funkcji konwerterów* i *funkcji walidatorów*, jeśli się różnią,
- zastosować właściwy dla dziedziny model *klas kształtów*,
- razem z *klasami kształtów* – dostarczyć ich reprezentacje graficzne.

Stopień elastyczności rozwiązania umożliwia pozostawienie wszelkich komponentów przetwarzających systemu bez zmian.

6.2.3 Implementacja komponentu przetwarzającego diagram kształtów regułami substytucji

Komponent został zaimplementowany w języku C++. Do wyboru tego konkretnego języka programowania nakłaniają następujące kwestie:

- kompilatory języka muszą generować wysokowydajny kod. Ma to szczególne znaczenie przy prowadzeniu przetwarzania *kształtów* w czasie rzeczywistym,
- kod źródłowy musi być przenośny, co otworzy dalsze możliwości przenoszenia eksperymentów na inne platformy,
- na rynku muszą istnieć interfejsy programistyczne do OpenGL dla tego języka,
- musi istnieć możliwość zintegrowania kodu źródłowego lub modułów skompilowanych z komponentami renderującymi *diagram kształtów,* wykorzystującymi rozszerzenia akcelerowane sprzętowo.

6.2.4 Implementacja komponentu przetwarzającego kształty w przestrzeni geometrycznej

Komponent ten został przygotowany w formie klasy zakodowanej także w języku C++. Udostępnia ona interfejs umożliwiający:

- wytypowanie *funkcji walidatora* na podstawie jego identyfikatora,
- przekazanie do funkcji meta-danych i zlecenie walidacji,

- wytypowanie *funkcji konwertera* na podstawie jego identyfikatora,
- przekazanie do funkcji meta-danych, określenie meta-danych danych tylko do odczytu i zlecenie przekształceń pozostałych danych.

Izolacja warstw przetwarzania powoduje, że jakiekolwiek aspekty ewentualnych przekształceń geometrycznych w ramach wykonywania ciał *funkcji walidatorów* i *funkcji konwerterów* są całkowicie niewidoczne poza komponentem (inny poziom abstrakcji). Dlatego informacje o przekształceniach nie są przekazywane przez interfejs i na zewnątrz.

Transformacje wyrażone treścią *funkcji konwerterów* lub *funkcji walidatorów* powinny operować w tzw. przestrzeni projekcji (*projective space)*[Har94]. Przestrzeń ta opisana może być dość dowolnie. Nawet konkretne wektory określające lokalizację można przetwarzać przy użyciu rozmaitych rachunków algebry liniowej. Po określeniu wymiaru przestrzeni projekcji powinno być możliwe zdefiniowanie transformacji. Na potrzeby eksperymentów zostały utworzony komponent posiadający funkcjonalność umożliwiającą:

- operowanie na wektorach przemieszczenia [p_x, p_y, p_z], skali [s_x, s_y, s_z] rotacji kątowej [r_x, r_y, r_z]. Implementuje on prosty rachunek wektorowy w przestrzeni trójwymiarowej Euklidesa. Meta-dane w przypadku tego komponentu są określone jako trzy trójki liczb zmiennoprzecinkowych. Funkcjonalność nie musi być rozbudowana i do celu prowadzenia podstawowych transfofmacji wystarczy umożliwiała:
  - przesunięcie o wektor (dodanie dwóch wektorów),
  - rotację o wektor zadany kątowo,
  - skalowanie (mnożenie wektora przez wartość skalarną),
  - obliczenie odległości w metryce euklidesowej.
- operowane na opisujących ransformacje macierzach projekcji [Har90], przygotowanych dla przestrzeni trójwymiarowej.

Z wykorzystaniem komponentu kodowane są *funkcje konwerterów* lub *funkcje walidatorów*.

Gdy informacja o lokalizacji ma postać macierzy, transformacja dokonywana jest poprzez standardowe mnożenie macierzy przez macierz kolejną transformacji. W wymiarze

praktycznym wyrażenie transformacji w postaci macierzy ma dość istotne znaczenie. Po pierwsze – umożliwi szybki proces łączenia transformacji geometrycznych, licznie występujących w *funkcjach konwerterów*. Ponadto - znacznie uprości projekcję perspektywiczną wyników, stosowaną obecnie na wielu platformach renderujących obrazy w grafice 3D. Wynik (wyrażony poprzez macierz) przed projekcją ulegnie jesze pomnożeniu przez tzw. macierz projekcji kamery [Har90]. W przypadku grafiki trójwymiarowej operacja ta przekształci obiekty opisane w przestrzeni trójwymiarowej w ich dwuwymiarowy rzut perspektywiczny na płaszczyznę projekcji, co umożliwia wyświetlanie na urządzeniach (2D).

Dla przestrzeni trójwymiarowej macierze transformacji będą miały wymiar 4x4 (rząd macierzy jest o jeden większy niż wymiar przestrzeni). Kształt macierzy transformacji zapożyczono z rozwiązań stosowanych w licznych bibliotakach graficznych, gdzie podstawą jakichkolwiek obliczeń w przestrzeni wirtualnej jest wspomniana macierz. Podstawowe macierze transformacji poglądowo przedstawia tabela.

Tabela 6.2: Przykładowe macierze (4x4) transformacji dla trójwymiarowej przestrzeni Euklidesa:

Rotacja wokół osi X o kąt α	Rotacja wokół osi Y o kąt α	Rotacja wokół osi Z o kąt α	Skalowanie proporcjonalne k	Przesunięcie o wektor [x,y,z]
$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) & 0 \\ 0 & \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \mathbf{1} & \mathbf{0} & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & \mathbf{1} \end{pmatrix}$

Zawartość Tabeli 6.2 jest ogólnie znana i omawiana w licznych pracach [Tch00]. Nie podlega tu rozważaniom. Jest cytowana wyłącznie w celach poglądowych (odwzorowanie meta-danych).

W implementacji komponentu do prowadzenia operacji na macierzach zastosowano funkcjonalność biblioteki graficznej OpenGL i dostarczane przez nią funkcje przetwarzające macierze transformacji. Biblioteka umożliwia także prowadzenie rachunku wektorowego. Jest to wystarczajace dla potrzeb eksperymentów. Macierz projekcji jest przechowywana w meta-danych *kształtu graficznego* jako 16 wartości zmiennoprzecinkowych, zapisywanych wierszami macierzy wierszami w kierunku od strony lewej do prawej. *Funkcje konwerterów* będą

korzystały funkcjonalności biblioteki graficznej OpenGL jako środka modyfikującego macierze opisujące geometryczne rozlokowanie *kształtów graficznych* w *diagramie kształtów*.

Do implementacji dodano także funkcje umożliwiające pobranie wektora pozycji, rotacji i skalowania z aktualnej macierzy projekcji dla danego *kształtu graficznego*. Są one wykorzystywane głównie przez *funkcje walidatorów* – operujące przeważnie na wektorowej postaci pozycji, rotacji czy wielkości (czy inaczej: skali) obiektu.

Reasumując – prowadzenie obliczeń dotyczących geometrycznego rozlokowania *kształtów* zarówno z użyciem rachuku wektorowego jak i i macierzowego ułatwi realizację eksperymentów na bazie systemu przetwarzającego *kształty graficzne*. Wykaże przy okazji możliwość swobodnego manipulowania architekturą warstwy przetwarzającej geometrię obiektów bez konieczności ingerowania w inne komponenty. Operowanie z użyciem tej samej wiedzy, ale nad różnymi przestrzeniami projekcji *kształtów* (2D, 3D) też będzie możliwe.

6.3 Implementacja narzędzia wizualizacji VisRuler

Transformacje 3D dokonywane z użyciem gramatyk K-GK można oceniać z wykorzystaniem aparatu matematycznego tylko w przypadkach prostych. Gdy gramatyka definiuje kilka reguł, zawiera kilkanaście symboli i relacji możliwa jest manualna predykcja efektów jej zastosowania (naturalnie warunkiem koniecznym jest tu znajomość początkowego *diagramu kształtów*). W przypadkach gramatyk bardziej skomplikowanych wymagane będzie użycie narzędzi umożliwiających wizualizację wygenerowanych wyników. Zadanie to spełnia pakiet graficzny VisRuler, stworzony przez autora. Zostanie on omówiony w niniejszym podrozdziale. Stanowi narzędzie wspierające testy systemów generacyjnych wykorzystujących gramatyki K-GK w niniejszej pracy.

6.3.1 Założenia wstępne

Prowadzenie licznych doświadczeń na materiale, którego właściwości mogą być oceniane jedynie w trybie analizy wizualnej spowodowało konieczność opracowania pakietu oprogramowania wspierającego trójwymiarową wizualizację *kształtów graficznych*. Standardowe narzędzia, umożliwiające renderowanie obiektów trójwymiarowych okazały się niewystarczające. Pozwalają one (jak Wings3D [Win09], XSI [Xsi10], Kinematix 3D Studio

[3ds10] etc.) jedynie na projekcję gotowej siatki 3D (lub zestawu siatek), będącego już końcowym wynikiem transformacji. Konieczne było opracowanie innego narzędzia, które po integracji z modułami wspierającymi transformacje 3D będzie w stanie w sposób przejrzysty prezentować dowolne wyniki eksperymentów. Także efekty prowadzenia poszczególnych kroków niejednokrotnie skomplikowanego procesu transformacji gramatyką K-GK. Istnieją już gotowe narzędzia wspierające wizualizację przetwarzania trywialnymi wariantami klasycznych gramatyk kształtu [Mcu90]. Są one jednak dedykowane do graficznej prezentacji wyników przetwarzania tylko konkretną gramatyką i jedynie uproszczonych strukturalnie *kształtów graficznych*. Nie spełniają zatem postawionych wymagań.

Głównym zadaniem programu VisRuler będzie wizualizacja wszelkich graficznych wyników testów prowadzonych na *diagramach kształtów* z wykorzystaniem gramatyki K-GK. Pracując w czasie rzeczywistym VisRuler będzie renderował komponenty umieszczając je w scenie 3D. Umożliwi graficzną prezentację komponentów modelu K-GK. W szczególności będą to:

- reprezentacje graficzne *kształtów*,
- symbole identyfikujące *klasy kształtu* K-GK,
- symbole identyfikujące relacje wiążące *klasy kształtu*, w przypadku obiektów wizualizowanych (reprezentacja graficzna w przestrzeni 3D).

VisRuler umożliwi prowadzenie procesu przetwarzania *diagramu kształtów* w trybie animacji, prezentując graficznie sekwencje transformacji dokonywanych na tym diagramie. Dostarczy także interfejsy użytkownika, umożliwiające swobodne przemieszczanie kamery po przestrzeni zawierającej reprezentacje graficzne *diagramu kształtów*.

6.3.2 Wybór platform i języków

Dla większości systemów doświadczalnych kluczowym atrybutem branym pod uwagę jest potencjalna przenośność i reużywalność komponentów powstającego rozwiązania. Stąd przesłanki do wyboru rozwiązań uniwersalnych - uniezależnionych od konkretnej platformy sprzętowej czy systemu operacyjnego. Jest oczywistym, iż tego typu podejście ma wiele zalet. Istnieją jednak sytuacje, gdzie warunkiem powodzenia implementacji jest utrzymanie należytej wydajności czy umożliwienie dostępu do konkretnych usług warstw niższych systemu

operacyjnego. W przypadku systemu prowadzącego w ramach eksperymentów wysokowydajne wizualizacje 3D w czasie rzeczywistym mamy do czynienia z taką właśnie sytuacją. Konieczność wykorzystania zaawansowanych funkcji akceleratora 3D (np. technologii vertex shaders) i trójwymiarowej bibliotegi graficznej skłoniła do zastosowania tu języka C++.

Naturalnie istnieją środki alternatywne (np. pakiet Java3D [Jav10] czy rozszerzenie OpenGL for Java [Jog10], jednak stopień ich zaawansowania i potencjalne możliwości wydajnościowe w ujęciu porównawczym są znacznie gorsze. Główną przyczyną takich niedogodności jest brak lub ograniczenie wsparcia sprzętowego dla technologii renderowania w czasie rzeczywistym (klatki 2D renderowane są programowo). Kwestie ograniczeń wydajnościowych dodatkowo uzupełnia brak potencjalnej możliwości wykorzystania języków programowania dla platform sprzętowych – ze sprzętowo wspartym przetwarzaniem vertex shaders na czele.

6.3.3 Silnik renderujący

Konieczność rozszerzania narzędzi przetwarzających elementy graficzne o możliwość ich wydajnej wizualizacji jest rzeczą oczywistą. W przypadku obecnego opracowania wizualizacja będzie elementem szczególnie ważnym. Natura przetwarzanego materiału jak i konieczność prowadzenia ciągłej interpretacji wyników eksperymentów przemawiają za zaangażowaniem w projekcie wysokowydajnego silnika graficznego. Silnik wspomagający wizualizację nastawiony będzie na akcelerowane sprzętowo renderowanie 3D w czasie rzeczywistym. Utrzymanie możliwości przenoszenia narzędzi na różne platformy sprzętowe wpłynie na jego architekturę. Wykorzystanie konkretnej biblioteki graficznej będzie możliwe tylko wtedy, gdy umożliwi ona renderowanie ramek sceny 3D w czasie rzeczywistym, będzie wspierana przez sprzęt rendrujący oraz będzie operowała na danych zgodnych z reprezentacją graficzną *kształtów* w wizualizowanym *diagramie kształtów*. Zakładając posiadanie wymaganych cech funkcjonalnych, najlepszą ocenę pod kątem przenośności uzyskuje obecnie biblioteka OpenGL. Ta biblioteka zostanie wykorzystana. Uzależnienie wszystkich eksperymentów od OpenGL nie jest jednak dokonane ostatecznie. Silnik renderujący jest także tylko jednym z komponentów graficznego systemu generacyjnego. Przekodowanie go do postaci wykorzystującej inną bibliotekę lub środowisko (na przykład Direct3D) nie stanowi większego problemu.

Wydajne renderowanie modeli 3D w czasie rzeczywistym wymaga licznych czynności przygotowawczych w fazie inicjalizacji pracy silnika. Do uzyskania wysokiego FPS (*Frames*

Per Second) konieczne jest zredukowanie ilości obliczeń prowadzonych przy renderowaniu każdej z klatek do niezbędnego minimum. Analiza siatek 3D i zlecanie rysowania każdego z ich elementów poprzez wywoływanie indywidualnych funkcji OpenGL jest wykluczone. Dlatego wszystkie modele 3D muszą być importowane z materiałów plikowych już w momencie inicjalizacji. Stanowiąc komponent niepodzielny, model może być „kompilowany" do postaci strumienia wielokątów i jednorazowo zapisany w pamięci akceleratora 3D. Kompilacja taka jest czynnością wspieraną przez OpenGL. Polega na zakodowaniu szeregu instrukcji OpenGL w postaci listy [Ogl10]. Instrukcje te mogą być rozkazami renderowania poszczególnych prymitywów, składających się na kształt siatki wielokątów, z których z kolei składa się model 3D. Podczas renderowania klatki animacji poszczególne elementy *diagramu kształtów* (czyli *kształty graficzne*) będą posiadały już swoje odpowiedniki (reprezentacje graficzne) w postaci strumieni wielokątów. Renderowanie sprowadzać się będzie do przesunięcia bieżącego punktu rysowania OpenGL w miejsce wskazane przez meta-dane *kształtu graficznego* i zlecenie w OpenGL jego renderowania (uruchomienie skompilowanej listy). Interfejs udostępniany przez silnik musi zatem umożliwić jednoznaczną identyfikację kompilowanej listy instrukcji na podstawie *klasy kształtu*. Samo utożsamienie reprezentacji graficznej *klasy kształtu* z materiałem plikowym (na przykład plikiem dyskowym zawierającym model 3D) można uzyskać choćby poprzez utrzymanie zbieżności nazw pliku i *klasy kształtu*.

Oprócz cech poza funkcjonalnych (wydajność renderowania i przenośność) silnik powinien posiadać liczne cechy funkcjonalne. Będą to:

- możliwość elastycznego sterowania projekcją 3D (kamerą), umożliwiając swobodną ekspozycję reprezentacji graficznej *diagramu kształtów*,
- możliwość nanoszenia komponentów dodatkowych (podziałka, raster, tło itp.),
- możliwość sterowania trybami projekcji sceny 3D (oświetlenie, koloryzacja, tryb *wireframe*, tryb przeźroczystości dla modeli 3D zanieczyszczających wyniki),
- możliwość eksportowania wyników (wyrenderowanych klatek) do zasobów plikowych,
- możliwość udostępnienia graficznego interfejsu wejściowego, także w projekcji innej niż projekcja 3D.

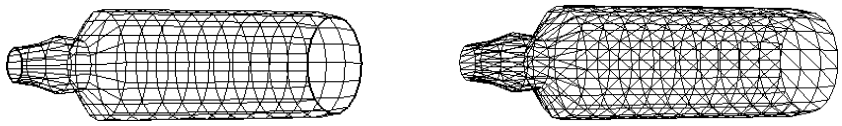
Istnienie komponentu spełniającego powyższe wymagania nie jest warunkiem krytycznym funkcjonowania systemu przetwarzającego z użyciem gramatyk K-GK. Na potrzeby oprogramowania symulacyjnego będącego przedmiotem testów w niniejszej pracy musiał jednak powstać. Bazują na nim narzędzia opisane w kolejnych podrozdziałach (Rysunek 6.9).

Rysunek 6.9: Okno graficzne renderowane przez silnik 3D autora podczas testów wydajności przy pełnej akceleracji sprzętowej (skomplikowane reprezentacje *kształtów graficznych* zawierające łącznie 2 870 000 trójkątów dały 89 FPS /*Frames Per Second*/ na sprzęcie Radeon HD 2400 XT) (źródło własne).

Zatem, w ramach realizacji wątku pobocznego zmierzającego do przygotowania oprogramowania symulacyjnego dla badań, autor stworzył profesjonalny, wysokowydajny silnik 3D bazujący na technologii OpenGL. Umożliwia on renderowanie kompletych scen 3D, mogących być podstawą gier 3D czy prezentacji interaktywnych. Drugą z przyczyn realizacji takiego zadania były podjęte w ramach pracy udane próby integracji narzędzi renderujących z technologiami Vertex Shaders (opisane w rozdziale 7.2.2).

Każda implementacja modułu wizualizacji jest uzależniona funkcjonalnie od *reprezentacji graficznej* przyjętej dla *kształtów* (zlokalizowanej na najniższym z trzech

poziomów abstrakcji). Na potrzeby eksperymentów przyjęto, iż moduł będzie operował na siatkach wielokątów w trówymiarowej perspektywicznej przestrzeni projekcji. Pozwoli to uzyskać przejrzyste wyniki symulacji i zwiększy poziom różnorodności możliwych do przeprowadzenia eksperymentów. Jak wiadomo – trójwymiarowa siatka wielokątów określana jest listą wierzchołków (verticles) zdefiniowanych w trójwymiarowej przestrzeni Euklidesa i relacjami zachodzącymi pomiędzy wierzchołkami. Wierzchołki znajdujące się na tej samej płaszczyźnie we wspomnianej przestrzeni mogą być łączone w listy cykliczne – definiując obwiednie wielokątów (faces). Zbiór wielokątów stanowi przedmiotową siatkę. Dla uzyskania lepszych parametrów wydajnościowych poprzez ułatwienie obliczeń siatkę taką poddaje się dodatkowo procesowi triangularyzacji (Rysunek 6.10), czyli konwersji (rozbicia) wszystkich wielokątów na trójkąty:



Rysunek 6.10: Wynik (po prawej) działania procesu triangularyzacji siatki czworokątów (po lewej) (źródło własne).

Tak przygotowana treść jest naturalnym materiałem renderowanym w czasie rzeczywistym przez wspomagane sprzętowo trójwymiarowe szybkie silniki graficzne.

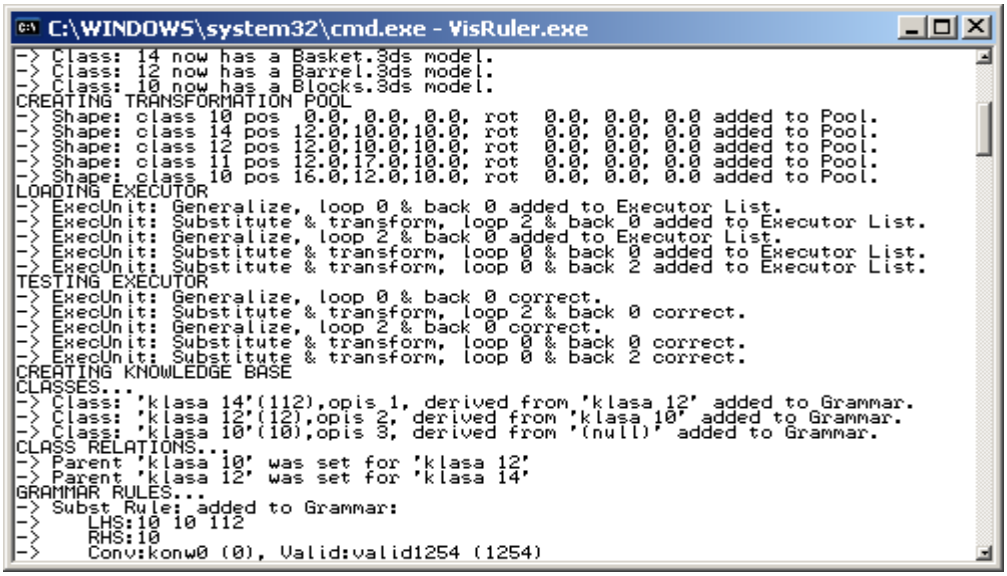
6.3.4 Komponenty narzędzia VisRuler

Moduł wizualizacji VisRuler został zakodowany w oparciu o bibliotekę graficzną OpenGL. Kod programu został zapisany w C++. Interfejsem okna graficznego jest GDI (co umożliwia łatwe przeniesienie kodu na większość platform – komputerowych systemów operacyjnych). Jak wspomniano - konieczne było napisanie kompletnego silnika 3D umożliwiającego wydajne renderowanie 3D przetwarzanych *kształtów*. Większość eksperymentów będzie produkowała wyniki w formie widoku 3D, często nie będące jedynie sceną statyczną lecz dynamicznie rozwijanym modelem. Kluczem do poprawnych interpretacji takich wyników jest posiadanie elastycznie sterowanego systemu wizualizacji, umożliwiającego ich szybką i rozbudowaną funkcjonalnie prezentację 3D.

Oprogramowanie składa się z następujących modułów:

- moduł wizualizujący - tworzy widok graficzny, przemieszcza kamerę, układa siatki 3D w przestrzeni, rysuje scenę z zastosowaniem silnika 3D,
- kontener dla siatek 3D - magazynuje siatki 3D będące odpowiednikami reprezentacji graficznych *kształtów* w *diagramie kształtów* - wymagany dla potrzeb wizualizacji,
- moduł interfejsu użytkownika - nakłada na scenę 3D widok pierwszoplanowy (2D), zwierający komponenty graficznego interfejsu użytkownika. Kolejkuje i obsługuje zdarzenia, dostarczane przez ten interfejs a generowane przez urządzenia wejścia,
- loader i pre-loader zasobów plikowych - w momencie uruchomienia systemu ładuje z materiałów plikowych wszelkie niezbędne komponenty, czyli:
  - wiedzę o *klasach kształtów,*
  - wiedzę o *regułach substytucji,*
  - początkowy *diagram kształtów,*
  - *diagram sterujący*,
  - reprezentacje graficzne *kształtów* (do pamięci oraz do pamięci akceleratora 3D przy pomocy silnika 3D).

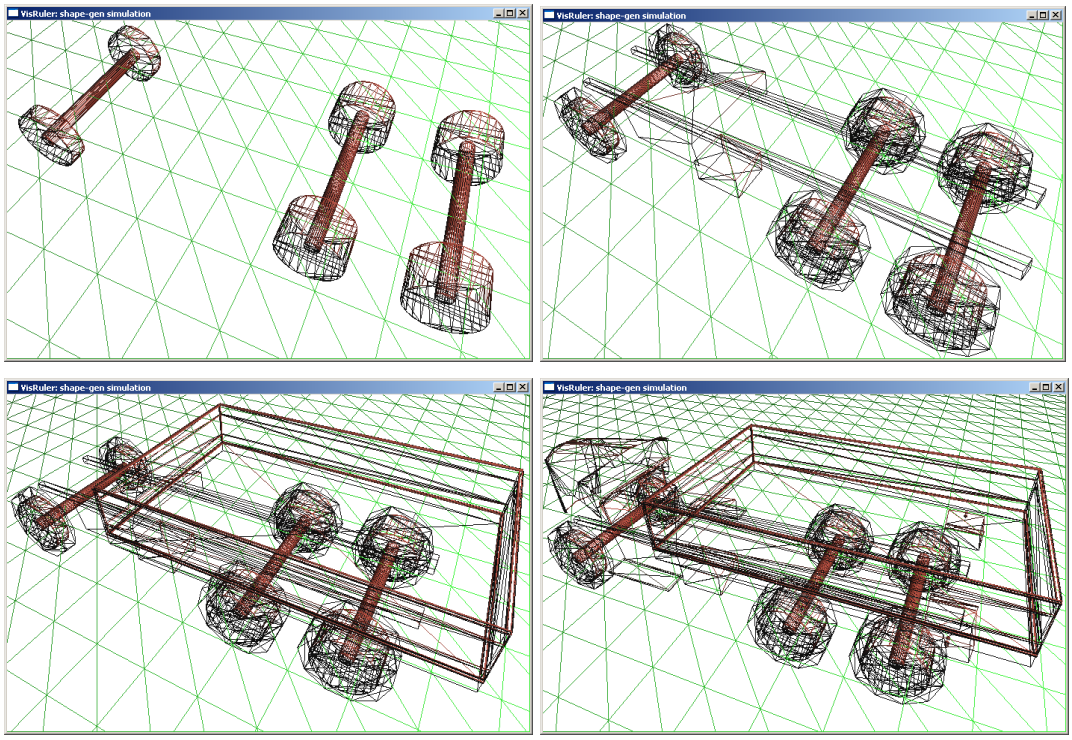
Moduł ładujący podczas pracy prowadzi walidację materiałów stanowiących wiedzę opisaną w języku CSGL. Umożliwia także łączenie wiedzy pochodzącej z różnych źródeł (Rysunek 6.11).



Rysunek 6.11: VisRuler, proces walidacji bazy wiedzy (źródło własne).

6.3.5 Sterowanie procesem przetwarzania kształtów

Komponentowa architektura implementowanego systemu umożliwiła łatwe sterowanie wszelkimi aspektami pracy modułów przetwarzających prowadzących transformacje K-GK w narzędziu VisRuler. Komponent przetwarzający *reguły substytucji* udostępnia swój interfejs bezpośrednio do rozważanego narzędzia. Wykonując operacje zakodowane w plikowej postaci *diagramu sterującego* umożliwia odseparowanie poszczególnych iteracji procesu przetwarzania *reguł substytucji* (Rysunek 6.12) i wizualizację międzywyniku.



Rysunek 6.12: VisRuler, wireframe, formy pośrednie przetwarzanego diagramu (źródło własne).

Daje to możliwość wymuszenia pracy krokowej narzędzia lub wprowadzania natychmiastowych korekt *diagramu sterującego* – zgodnie z podziałem narzucanym przez instrukcje tego diagramu.

6.3.6 Zastosowania narzędzia VisRuler

VisRuler odegrał znaczącą rolę podczas prac nad ustabilizowaniem definicji modelu K-GK. Umożliwił przeprowadzanie szybkich testów wszelkich weryfikowanych rozwiązań, dotyczących samej koncepcji przetwarzania generacyjnego z zastosowaniem struktur hierarchicznych.. Dzięki dodaniu procesu automatycznego ładowania zasobów podczas startu możliwe było skrócenie do minimum nakładów czasowych wymaganych dla skonfigurowania pojedynczego eksperymentu. Będąc oprogramowaniem zrealizowanym przez autora VisRuler był konsekwentnie rozbudowywany i mógł sprostać potrzebom wizualizacji kolejnych materiałów graficznych, zadawanych pod kolejnymi postaciami i przeprowadzaniu kolejnych eksperymentów.

7 Badania właściwości zrealizowanych rozwiązań

W niniejszym rozdziale opisane zostały próby wykorzystania przedstawionej w pracy metodologii do implementacji graficznych systemów generacyjnych. Realizując takie systemy brano pod uwagę nastąpujace zagadnienia:

- problem generowania komponentów obiektów graficznych na podstawie ich geometrycznych właściwości oraz na podstawie wiedzy o jakościowej klasyfikacji tych komponentów. Podjęcie tego zagadnienia miało na celu ustabilizowanie technik projektowania gramatyk K-GK w prostych systemach generacyjnych. Rozwiązanie wymagało opracowania wytycznych dla procesu formatowania rejestru *klas kształtów*, wytworzania odpowiedników ontologicznych tych klas, oraz (w ujęciu wielowarstwowym) tworzenia list *reguł substytucji*.

- zagadnienie sekwencyjnego sterowania obiektami w przestrzeni trójwymiarowej. Postawione tu zadanie polegać będzie na opracowaniu rozwiązań umożliwiających przygotowywanie gramatyk iteracyjnie modyfikujących geometryczne właściwości tych samych obiektów. Przeznaczeniem takiego rozwiązania będzie wspomaganie animacji w trójwymiarowych prezentacjach renderowanych w czasie rzeczywistym. Poszczególne rozwiązania wykazały sporą elastyczność proponowanego modelu przetwarzania także w takich zastosowaniach. Z myślą o tej grupie zastosowań zostały przeprowadzone testy wydajności implementacji algorytmów wyszukujących reguły w bazie wiedzy oraz dokonujących na ich podstawie czynności edycyjnych. Wyniki przedstawiono w rozdziale 7.2.1.

- problem umożliwienia integracji sprzętowo realizowanych warstw modelujących geometrię materiałów 3D z warstwami modelu K-GK. Rozwiązanie tego problemu stanowiło najbardziej dobitny dowód wysokiej re-używalności i elastyczności systemów graficzych tworzonych w oparciu o proponowaną metodykę. Technologię wspomagającą sprzętowe renderowanie 3D w czasie rzeczywistym udostępniają warstwy modyfikujące wierzchołki wielokątów z potoku graficznego akceleratora 3D bezpośrednio przed procesem rasteryzacji. Stworzone rozwiązanie udowadnia, iż możliwe jest ścisłe zintegrowanie warstw budowanego systemu z warstwami zarezerwowanymi wcześniej jedynie dla takich operacji jak na przykład

renderowanie efektów specjalnych (w już przygotowanym materiale) – długo po zakończeniu geometrycznych obliczeń na nim prowadzonych. W tym konkretnym przypadku wpłynie to na znaczne podniesienie wydajności prowadzonych w systemie graficznym obliczeń.

Większość badań wymaga wykorzystania komponentów zaimplementowanego systemu przetwarzającego *kształty graficzne* zgodnie z wytycznymi metodyki przetwarzania *kształtów* w gramatykach K-GK. Każdorazowo licznych modyfikacji będzie wymagała przetwarzana przez system gramatyka. Niekiedy – także implementacja komponentów systemu. Analiza własności tworzonych graficznych systemów generacyjnych pozielona została na dwa etapy:

- badanie własności poza funkcjonalnych zrealizowanego przykładowego systemu generacyjnego,
- wykazanie istnienia cech funkcjonalnych graficznych systemów generacyjnych, których pojawianie się jest powodowane definiowaniem w tych systemach odpowienio wyspecjalizowanych gramatyk K-GK.

Wykazanie własności poza-funkcjonalnych badanego systemu nie przysparza trudności. Polegało na prowadzeniu testów wydajności systemu przetwarzającego *kształty graficzne*, oraz rozważaniach dotyczących architektury systemu w kontekście jego przenośności czy łatwości wykorzystania elementów. Cześć tych rozważań była już przedmiotem poprzedniego rozdziału.

Badanie potencjalnych cech funkcjonalnych projektowanych systemów uzależnione jest głównie do treści gramatyki K-GK oraz sposobu wykorzystywania komponentów przetwarzających *kształty graficzne*. Dotyczy konkretnych zagadnień, rozważanych w ramach rozdziału 7.1.

7.1 Analiza właściwości funkcjonalnych systemu

Niniejszy rozdział podzielono na dwie indywidualne części zgodnie z dwoma odkrytymi obecnie nurtami zastosowań systemów opartych na K-GK. Są to:

- wykorzystanie go jako klasycznego systemu generacyjnego w rozwijaniu uporządkowanych strukturalnie graficznych obiektów trójwymiarowych,

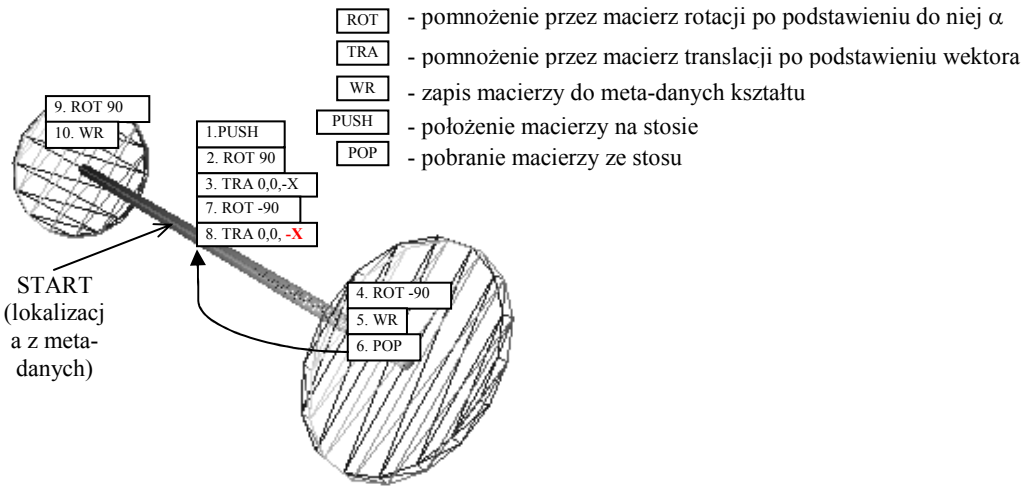
- wykorzystanie go w zastosowaniach do sterowania obiektami graficznymi w wyizolowanej trójwymiarowej przestrzeni projekcji z renderowaniem z czasie rzeczywistym.

7.1.1 Wspomaganie przetwarzania modeli statycznych

Sterowanie procesem generowania kształtów

Pierwszy test sprawdza funkcjonowanie wyizolowanych *gramatyk kształtu* osadzonych w modelu K-GK. Zgodnie z założeniami modelu - gramatyki te operują na materiale, którego forma została już jakościowo zidentyfikowana. Tym samym będzie możliwe dostarczenie transformacji, które umożliwią wyizolowanie pod-kształtu z przetwarzanego materiału. Jest to konieczne do zastosowania reguły kształtu. W przypadku banalnym – zbiór tranzycji gramatyki kształtu będzie zawierał tylko jeden element: tranzycję *kształtu* początkowego do modelu 3D, odpowiadającego wizualnie *klasie kształtu*. Model ten stanie się tym samym jego reprezentacją graficzną i będzie renderowany.

To jednak dopiero ostatni etap przetwarzania. Jak wiadomo *reguły substytucji* K-GK zawierają *funkcje konwerterów*. Operacje geometryczne powodowane przez te funkcje razem z treścią samych reguł są narzędziem modyfikującym *diagram kształtów*. Gdy diagram ten będzie zawierał wyłącznie *kształty* terminalne – zostaną one przetworzone zwykłymi gramatykami (zgodnie z końcowym przyporządkowaniem do klas). W doświadczeniu operacje wykonywane przez *funkcję konwertera* opierają się na transformacjach macierzy projekcji w *przestrzeni projekcji*. Zarówno rotacja, skalowanie jak i przesunięcie o wektor w przestrzeni 3D będą realizowane z użyciem macierzy transformacji przedstawionych w rozdziale 6.2.4. Pomnożenie bieżącej macierzy transformacji przez kolejną macierz spowoduje wygenerowanie złożenia tych transformacji. Warto zauważyć, iż przy wykorzystaniu rachunku macierzy transformacji nie jest konieczne interpretowanie globalnych współrzędnych modyfikowanego *kształtu*. Dlatego projektowanie *funkcji konwertera* będzie każdorawowo polegało na interpretowaniu bieżącego usytuowania w przestrzeni trójwymiarowej, określonego macierzą. Dodatkową zaletą takiego podejścia jest możliwość łatwego archiwizowania pozycji tymczasowych w tej przestrzeni. Wykorzystano do tego celu stos macierzy dostarczany przez bibliotekę graficzną. Na rysunku 7.1 przedstawiono operacje wykonywane przez funkcję konwertera w celu określenia lokalizacji dwóch nowych obiektów (tu: kół).



Rysunek 7.1: Operacje wykonywane przez funkcję konwertera bazującą na przetwarzaniu macierzy transformacji. Przykład konfiguracji dwóch nowych *kształtów* (połączonych „kół" agregowanych w późniejszym modelu) (źródło własne).

Funkcja konwertera z powyższego przykładu kodowana z zamiarem wykorzystania jej w bibliotece graficznej (na przykład OpenGL) będzie zatem określana następujacycm pseudokodem:

```
MetaData startData ReadMetaData(KsztaltStart)
PushMatrix();
Rotate (90,0,1,0);  // parametry to kąt rotacji oraz mnożniki rotacji kątowej wokół osi x,y,z
Translate (0,0,-startData.scale.X);
Rotate (-90,0,1,0);
WriteMetaData (KsztaltKolo1);
PopMatrix();
Rotate (-90,0,1,0);
Translate (0,0,-startData.scale.X);
Rotate (90,0,1,0);
WriteMetaData (KsztaltKolo1);
```

Jak wiadomo lista *reguł substytucji* umożliwia także sterowanie procesami rozbudowy *kształtu graficznego*. Możliwe jest tworzenie list *reguł substytucji*, które gwarantują jednotorowy przebieg procesu tworzenia *kształtu* bez konieczności ingerowania w jego etapy *diagramem sterującym*. Wówczas *diagram sterujący* zawierał będzie wyłącznie wykonywany w pętli rozkaz uruchomienia kolejnej iteracji procesu (bez potrzeby wymuszania generalizacji jakichkolwiek *kształtów graficznych* czy kończenia całego procesu). Osiągnięcie w *diagramie*

kształtów wyłącznie *kształtów* terminalnych będzie gwarantowana treścią poszczególnych *reguł substytucji*.

Uwaga: Dla właściwej interpretacji tej jak i następnych gramatyk może być przydatne przypomnienie cechy procesu przetwarzania K-GK, wywodzącej się z definicji algorytmu dopasowywania *reguły substytucji* (rozdział 5.4.10):

W każdej iteracji kształt należący do diagramu kształtów może być dopasowany tylko raz, niezależnie od tego czy ulegał wcześniej transformacji lub czy był jedynie kontekstem przy dopasowywaniu innej reguły substytucji.

Przyjmując wyżej opisane założenia przeprowadzono próbę stworzenia gramatyki K-GK, generującej układ brył geometrycznych na podstawie ich powiązań. W tym celu:

1. Wybrano dziedzinę: [obiekty mobilne]/pojazd/podwozie zawierającą zbiór klas C_A.
2. Zdefiniowano początkowy *diagram kształtów* jako D.
3. Do *klas nie abstrakcyjnych* dodano gramatyki kształtu, których zbiory reguł kształtu zawierają jedynie regułę przeprowadzającą kształt początkowy w kształt terminalny będący siatką wielokątów (reprezentacją graficzną kształtu). Zakłada się tu naturalnie, że terminy „kształt początkowy" w „kształt terminalny" są wyrażone w rozumieniu klasycznej gramatyki kształtu.
4. Zdefiniowano listę R *reguł substytucji* i klas będzie pochodził gramatyki K-GK, która służy do opisu podwozia pojazdu. Zbiór reguł będzie generował elementy „podwozia trójosiowej ciężarówki":

C_A = {abstract P (Punkt), abstract Po (PunktOsi), abstract Lo (LiniaOsi), Ot (OsTyl), Op (OsPrzod), (Kp) KoloPrzod, (Kt) KoloTyl}

D = {P}

R = {

(1) P + Po + Po + Po –=> Po + Po + Po, TRUE , NULL);

// redukcja wykorzystanego już kształtu Punkt

(2) P +=> P + Po + Po + Po , TRUE, kon2 (IN P, OUT Po, OUT Po, OUT Po);
// utworzenie trzech osi na podstawie kształtu początkowego

(3) Po + Po + Po :=> Op + Po + Po, wal3 (IN Po + IN Po + IN Po), NULL;
// wyspecyfikowanie klasy osi przedniej

(4) Po :=> Ot, NULL, NULL;
// wyspecyfikowanie klas osi tylniej (ta reguła zadziała dwa razy)

(5) Op :=> ~Op + Kp + Kp, wal5 (IN Op), kon5(IN Op, OUT Kp, OUT Kp)
// dodanie kształtów kół do przedniej osi

(6) Ot :=> ~Ot + ~Kt + ~Kt, wal5 (IN Ot), kon5(IN Ot, OUT Kt, OUT Kt)
// jak poprzednio, lecz dla tylnej osi (ta reguła wykona się 2 razy)

}

5. Zdefiniowano *funkcje konwerterów* i *funkcje walidatorów* do takiej gramatyki, zakodowane w trójwymiarowej przestrzeni Euklidesa. Będą one zawierać ransformacje geometryczne, pełniące następujące zadania:

kon2 (IN, OUT, OUT, OUT) – ustalenie pozycji osi w pojeździe,

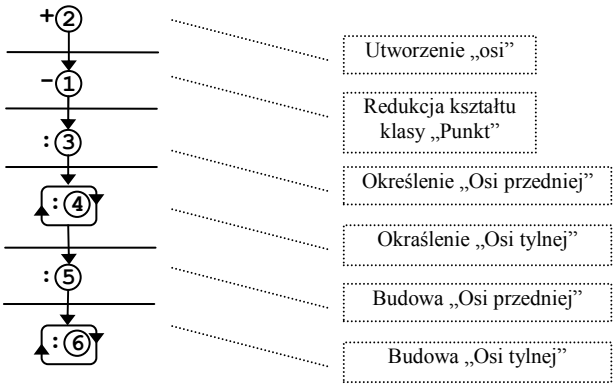
wal3 (IN, IN, IN) – sprawdzenie, czy mamy do czynienia z prawidłowym pojazdem,

wal5 (IN) – sprawdzenie prawidłowego ustawienia osi (dystans od podłoża). Funkcja Wal5 (IN) będzie także wykorzystywana przez regułę (6),

kon5(IN, OUT, OUT) - ustalenie pozycji kół na podstawie cech geometrycznych osi,

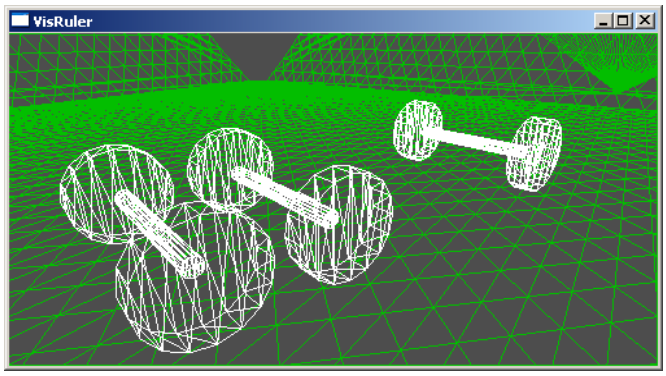
NULL – brak w regule funkcji operującej na geometrycznej lokalizacji *kształtu*.

Diagram K-GK (opracowany w ramach przedmiotu pracy i opisany w rozdziale 5.6.4) będzie dla powyższej transformacji wyglądał jak na rysunku 7.2.



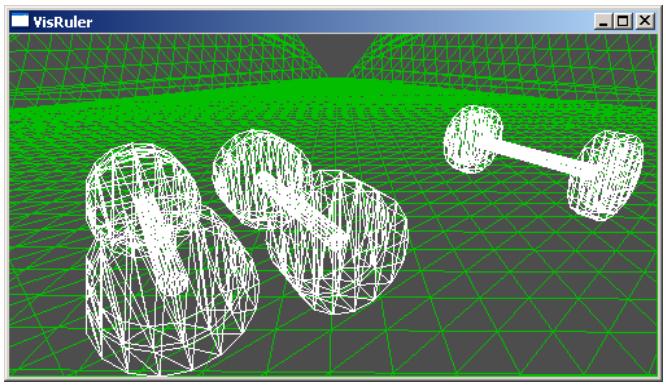
Rysunek 7.2: Diagram K-GK wykonania *reguł substytucji* (1) – (6) (źródło własne).

Warto zauważyć, ze funkcje wal5(), kon5() zostały zapożyczone do reguły (6), gdyż wykonują dokładnie takie same funkcje kontrolne i walidacyjne. Efekt wykonania sześciu reguł ostał przedstawiony na rysunku 7.3.



Rysunek 7.3: Efekt wykonania *reguł substytucji* (1) – (6). Reprezentacje graficzne *kształtów* wygenerowane są gramatykami kształtu *klas kształtu* Kt, Kp, Ot, Op (źródło własne).

Ponadto zależnie od rozmieszczenia gramatyk w *klasach kształtu* osiągamy zupełnie różną reprezentację graficzną końcowego *diagramu kształtów* w przestrzeni projekcji. W powyższym przykładzie gramatyki kształtu dla Kt i Kp są identyczne, podobnie jak dla Ot i Op. Przypisanie w modelu K-GK *gramatyki kształtu* do *klasy kształtu,* a nie *kształtu graficznego* pozwala na sprawne manipulowanie reprezentacjami graficznymi komponentów *diagramu kształtów*. Jedna podmiana gramatyki kształtu (w *klasie kształtu* Kt) dała efekt przedstawiony na rysunku 7.4.



Rysunek 7.4: Modyfikacja reprezentacji graficznej poprzedniego wyniku przy użyciu gramatyk kształtu (źródło własne).

Uruchomienie wyżej przedstawionych przykładów gramatyk zademonstrowało zalety warstwowego modelu przetwarzania *kształtów*. Nie interpretując *gramatyką kształtu* lokalizacji

kształtu w globalnej przestrzeni projekcji jesteśmy w stanie przetwarzać nią pod-kształt sklasyfikowany. Transformacja ta będzie przebiegała wielokrotnie, gdyż dla każdego wystąpienia *kształtu graficznego* danej *klasy kształtu* na *diagramie kształtów*. Komponent zarządzający *funkcjami walidatorów* i *funkcjami konwerterów* wymusi właściwe usytuowanie *kształtu* w globalnej przestrzeni projekcji. Gramatyk generujących *kształty* do tej samej przestrzeni i w tak uporządkowany sposób może być wiele.

Zbiory R i C_A umieszczone w przykładzie można dalej rozwijać nie przeprowadzając od razu *kształtów* Kt, Kp, Ot, Op do postaci terminalnej, lecz rozbudowując je do zawierającej kolejne elementy szczegółowe. Widać tym samym jak przy użyciu listy *reguł substytucji* wprowadzających uporządkowanie jakościowe elementów, oraz ich *funkcji konwerterów* wprowadzających uporządkowanie geometryczne możliwe jest wyrażenie relacji typu whole-part (całość-cześć) między fizycznymi komponentami rzeczywistego obiektu. W konsekwencji posiadania tej wiedzy – możliwe jest generowanie precyzyjnie określonych nowych *kształtów graficznych*.

Zagadnienie generowania fraktali, jako złożonych kształtów graficznych

Problem komponowania fraktali na bazie *kształtów* samo-podobnych jest powszechny w opracowaniach wspomagających projektowanie czy modelowanie w grafice. W niniejszym podrozdziale zostanie opisana implementacja procedury generującej takie konstrukcje w gramatyce K-GK. Rozdzielenie abstrakcji sklasyfikowanego *kształtu graficznego* od jego reprezentacji graficznej daje jak wiadomo możliwość rozbudowy struktury modelu jeszcze bez reprezentowania go graficznie. Wprowadzona z modelu K-GK klasyfikacja *kształtów* pozwala na umieszczanie w *diagramie kształtów* elementów, które będą *kształtami* lecz nie będą komponentami widocznymi w przestrzeni projekcji. Są *kształtami klas abstrakcyjnych.* W poruszanym zagadnieniu daje to możliwość zbudowania systemu kontroli przetwarzania *kształtów* w oparciu o specjalne *kształty*-znaczniki. Ich obecność w *diagramie kształtów* będzie powodować uruchomienie kolejnej iteracji prowadzonych transformacji - określając w konsekwencji miejsce ulokowania obiektu, czy narzucając ograniczenia ilościowe tworzonych obiektów potomnych.

Procedura generująca będzie rozwijała *diagram kształtów* w kierunku „od ogółu do szczegółu”. Umieszczenie *reguł substytucji*:

(1) z0 + z1 + z1 + z1 –=> z1 + z1 + z1, w1(z0), NULL

(2) z0 +=> z0 + z1 + z1 + z1, w2(z0), k2(z0,z1,z1,z1)

Spowoduje utworzenie trzech znaczników z1 w miejscach obliczonych przez k(z0,z1,z1,z1) na podstawie meta-danych z0 i usunięcie niepotrzebnego już z0. Następnie zapisanie:

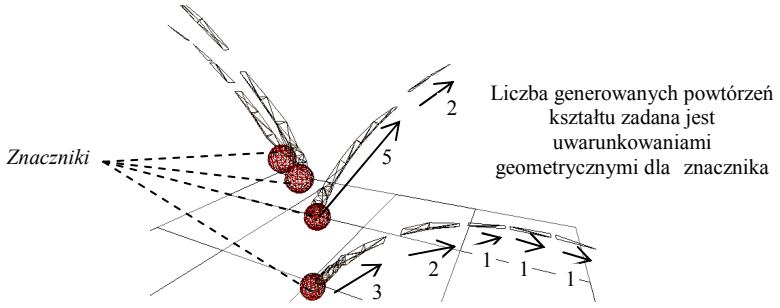
(3) z1 + k1 :=> z1 + ~k1, NULL, NULL

(4) z1 +=> z1 + k1, w4(z1), k4(z1, k1)

będzie powodowało tworzenie kolejnych elementów serii zgodnie w ilości z treścią w4(z0) jednocześnie modyfikując funkcją k4(z1, k1) meta-dane zarówno z1 jak i k1. Te meta-dane i *funkcja konwertera* nad nimi wykonywana określą ile instancji k1 zostanie wygenerowane. W momencie wyczerpania się możliwości wykorzystania któregokolwiek znacznika z1 proces przejdzie do kolejnej reguły. I tak do momentu wyczerpania możliwości wykorzystania z1 w LHS reguły (3). Wówczas można dodać regułę kasującą:

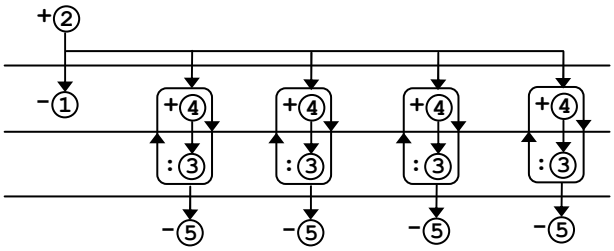
(5) z1 –=> , w4(z1), NULL

Powyższa konstrukcja umożliwiła zaimplementowanie pętli generacyjnej z możliwością korygowania meta-danych każdego z tworzonych *kształtów*. Pozwoliła na iteracyjne generowanie kolejnych komponentów modelu z utrzymaniem możliwości precyzyjnego uwarunkowania liczby iteracji i na podstawie geometrycznego rozlokowania *kształtów* sterujących zapisanego w meta-danych (Rysunek 7.5).



Rysunek 7.5: Znaczniki sterujące i wpływ ich usytuowania na wykonanie dalszych *reguł substytucji* (źródło własne).

Diagram K-GK dla tak zaproponowanej konstrukcji będzie tym razem zorientowane wokół znaczników zależności pomiędzy uruchamianymi regułami. Każdy znacznik stworzy indywidualną i odseparowaną serię takich zależności – w której uruchomienie danej reguły będzie zależało od wyniku uruchomienia innej. Diagram przedstawiono na rysunku 7.6.



Rysunek 7.6: Diagram K-GK przetwarzania znaczników sterujących powyższymi *regułami substytucji* (źródło własne).

Generowanie serii obiektów o identycznym lub zbliżonym kształcie okazuje się łatwe do wykonania po wprowadzeniu do *diagramu kształtów* abstrakcyjnych *kształtów* kontrolnych (znaczników). W swoich meta-danych posiadają one dodatkowo informację o wektorze skali i rotacji (Rysunek 7.7).

Rysunek 7.7: *Kształty* terminalne wykorzystanie do tworzenia omawianego modelu z elementami samo-podobnymi (po lewej), oraz oznakowanie *kształtu* abstrakcyjnego (znacznika) – stosowane na dalszych rysunkach (źródło własne).

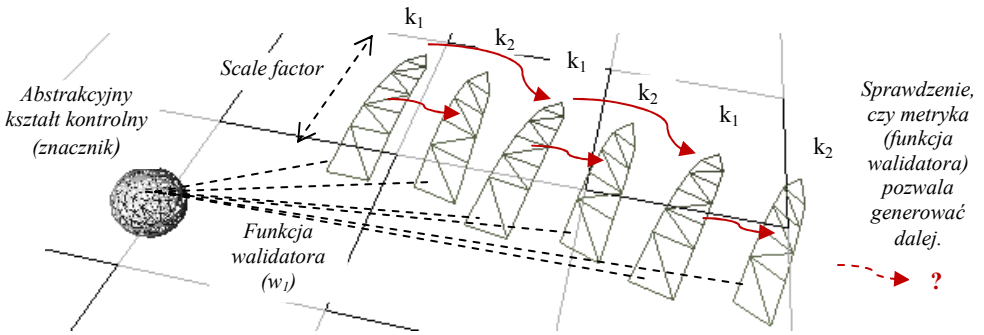
Generowanie serii elementów można w kontrolowany sposób prowadzić przy pomocy tylko jednej *reguły substytucji*. Okazuje się, że nie destabilizując procesu przetwarzającego iterację, można poszerzać zawartość listy *reguł substytucji* o dodatkowe reguły generujące odseparowane i pojedyncze *kształty*. Wystarczy umieścić:

```
(2a)    z1 + k1 + k1 :=> z1 + k1 + ~k2, w2a(z, k1,k1), r2a(k2)
```

oraz generować jednoześnie dwa nowe *kształty* `k1` regułą `(4)`. Jej nowa wersja to:

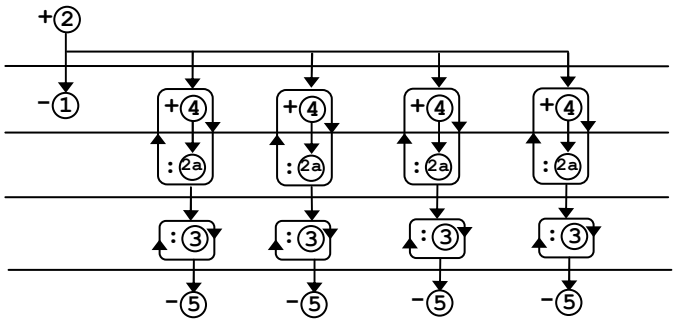
```
(4)     z1 +=>  z1 + k1 + k1, w4(z1), k4(z1, k1, k1)
```

Reguła substytucji `2a` przeprowadzi parę *kształtów* `(k1 + k1)` w `(k1 + k2)` uzależniając wykonanie tej transformacji od prawidłowego usytuowania geometrycznego `w2a(z1, k1, k1)`. Ponieważ meta-dane `k1` przeprowadzonego w `k2` są już wypełnione *reguła* `(2a)` dokonuje wyłącznie konwersji *klasy kształtu* nie zmianiając już treści meta-danych (Rysunek 7.8).



Rysunek 7.8: Generowanie serii komponentów k_1+k_2 każdorazowo uzależniane od wyniku *funkcji walidatora* w_1. Skalowanie lub inne operacje geometryczne na *kształtach* wymuszane są treścią *funkcji konwertewów reguł substytucji* (źródło własne).

Kolejnym bardzo istotnym aspektem omawianej konstrukcji jest uzależnianie generowanie *kształtów* `k1` lub `k2` od obecności `z1`. Teoretycznie wyprowadzenie z `(k1+k1)` *kształtów* `(k1+k2)` byłoby możliwe i bez udziału `z1`. Zastosowana konstrukcja powoduje jednak, że `z1` jest aktywatorem jakichkolwiek operacji nad serią *kształtów* `(k1+k2)`. Jego usunięcie powoduje zamknięcie generowania serii. Diagram K-GK po modyfikacjach będzie miał kształt prezentowany na rysunku 7.9.



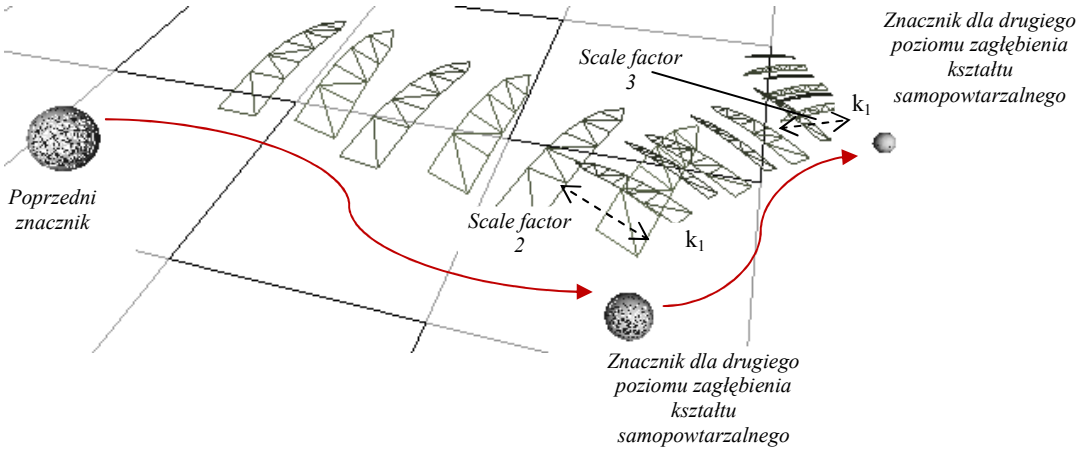
Rysunek 7.9: Diagram K-GK generowania serii komponentów k_1+k_2 z rysunku 7.8 (źródło własne).

W podjętych próbach zdefiniowania gramatyki generującej *kształty* samo-podobne starano się redukować liczbę koniecznych do rozważenia *reguł substytucji* oraz liczbę znaczników znajdujących się jednocześnie na *diagramie kształtów*. Najlepszy wynik uzyskano generując następny znacznik kontrolny dopiero po wygenerowaniu całej bieżącej serii elementów. Głównym problemem przy projektowaniu gramatyki było kontrolowanie warunku zakończenia rozwijania modelu po osiągnięciu odpowiedniego poziomu zagłębienia serii *kształtów* (a tym samym warunku stopu procedury). Gorsze rezultaty dało definiowanie nowych *klas kształtów* dla każdego z rzędów (konieczna większa liczba reguł generujących). Bardziej

wartościowe okazało się ponowne wykorzystanie cechy modelu K-GK polegającej na uzależnieniu uruchomienia *reguł substytucji* od własności geometrycznych znacznika:

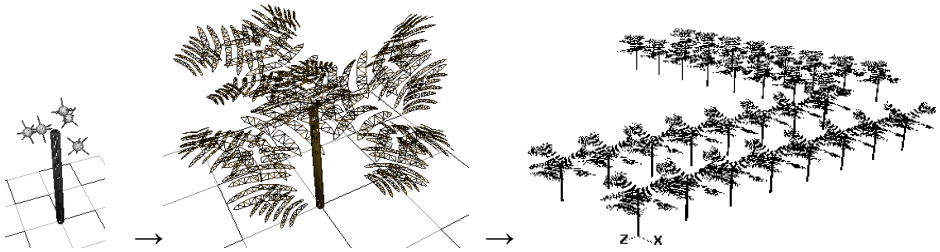
(4a) z1 :=> z1, w4a(z1), k4a(z1)

Powyższa reguła zostanie uruchomiona w momencie, gdy z1 nie zostanie dopasowany do reguły (4) więc nie jest już potrzebny. Zastąpi go nowy *kształt* kontrolny z1. *Funkcja walidatora* w4a(z1) posłuży do sprawdzenia możliwości wykonania operacji przeniesienia znacznika z1. Jeśli taka operacja nie jest już możliwa, zadziała *reguła substytucji* (5) i usunie znacznik. Cechy geometryczne nowego *znacznika* są modyfikowane w momencie rozpoczynania wykonania serii definiuje k4a(z1) (Rysunek 7.10).



Rysunek 7.10: *Kształty* samo-podobne tworzone *regułami substytucji*.

W przypadku badanego systemu bezproblemowe staje się także realizowanie transformacji hybrydowych – które zależnie od treści *reguł substytucji* rozbudują *kształt* samo-podobny lub zwykły. Na *diagramie kształtów* znaczników z1 może być dowolnie dużo. Każdy posiadając indywidualnie zdefiniowane meta-dane, zostanie przetworzony zaproponowanymi regułami generując *kształt* samo-podobny. *Diagramy kształtów* zawierać będą *kształty* samo-podobne jako jedne z wielu innych komponentów. Dzięki wprowadzeniu jednoznacznej identyfikacji *klasy kształtu* rozlokowanie geometryczne *kształtów* samo-podobnych możemy reorganizować poprzez wymianę *reguły substytucji* (2) – na zawierającą inną *funkcją konwertera* czyli generującą inne usytuowanie znaczników z1. Zwielokrotnienie operacji definiowanej regułą (2) następuje poprzez dodanie analogicznej grupy reguł ustawiających znaczniki z0 w szyku liniowym (Rysunek 7.11).



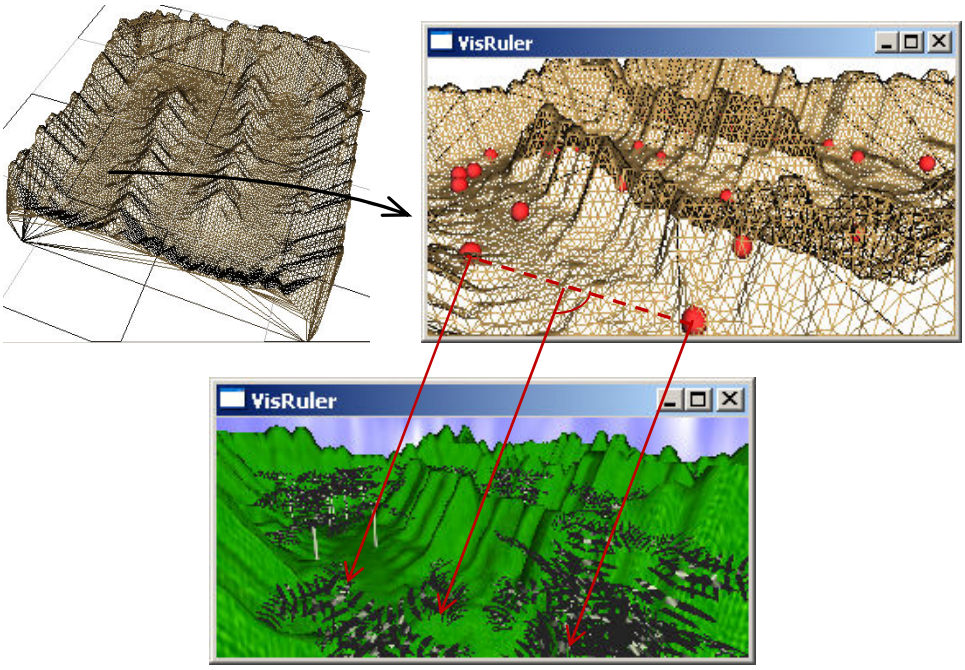
Rysunek 7.11: Wynik uruchomienia reguły (2) (po lewej), rozwinięcie gramatyką kształtu z_0 gdzie sam z_0 przechodzi w graficzny „słupek" (pośrodku), zwielokrotnienie wyniku doświadczenia w kolejnej serii danej na bazie znacznika z_0 (po prawej) (źródło własne).

Efekt dowodzi dużej elastyczności w formułowaniu zachowania systemu generacyjnego. Oddzielenie abstrakcji *kształtu graficznego* od jego reprezentacji graficznej daje możliwość sprawnego formułowania transformacji bazując na modelu logicznym, a kontrolując ich przebieg poprzez stawianie uwarunkowań natury geometrycznej dla konkretnych *kształtów*.

Idąc dalej przyjętą ścieżką projektową podjęto próbę rozpatrzenia zagadnienia inżynieryjnego, polegającego na wygenerowaniu kompletnej zawartości przestrzeni projekcji z zastosowaniem przyjętej metodyki i wcześniej omówionych rozwiązań szczegółowych. Gotowy produkt, dostarczony w ramach zagadnienia, jest materiałem umożliwiającym wyrenderowanie wirtualnej przestrzeni gry 3D lub scenerii interaktywnej trójwymiarowej prezentacji. Zagadnienie to sprowadza się do wypełnienia wygenerowanymi komponentami przestrzeni projekcji. Przestrzeń ta jest często geometrycznie ograniczona rozmaicie ukształtowanym „podłożem", do którego trzeba adaptować wszelkie nanoszone obiekty graficzne. Materiałem dostarczonym (wejściowym) są tu:

- opisane modelem 3D graficzne „podłoże", będące wytekstu rowaną i najczęściej silnie zdeformowaną siatką wielokątów. Siatka ta definiuje ograniczenia w rozlokowaniu obiektów dodawanych,
- znaczniki topologii terenu, będące *kształtami*. Znaczniki te umożliwiają adaptację nanoszonych obiektów do topologii terenu,
- zbiór *reguł substytucji* określających transformacje konstruujące poszczególne obiekty,
- *kształty* terminalne, będące trójwymiarowymi siatkami wielokątów.

Duża ilość angażowanych na takiej scenie elementów oraz widoczna niejednokrotnie hierarchia tych elementów to przesłanki do wykorzystania testowanej metodyki również w takim zagadnieniu. Na podstawie znaczników topologii siatki generowane są komponenty przestrzeni projekcji (Rysunek 7.12). *Funkcja konwertera* określa składową Y pozycji nanoszonego *kształtu* prowadząc intepolację liniową pozycji Y wybranych znaczników. *Reguły substytucji* określają jakiej *klasy* znaczniki mogą stanowić podstawę do wygenerowania nowego *kształtu graficznego* w *diagramie kształtów*. Zawarte w nich *funkcje walidatora* określają geometryczne uwarunkowania dla tych *kształtów*.



Rysunek 7.12: Etapy tworzenia prezentacji trójwyiarowej opartej o siatkę terenu i znaczniki jego topologii. Na skonstruowanej dla potrzeb doświadczenia siatce widoczne są znaczniki topologii terenu i „potomne" *kształty graficzne* (źródło własne).

Algorytm generowania dalszych znaczników na podstawie istniejących już znaczników topologii terenu jest tu dowolny. Zależeć będzie od dziedziny zagadnień, dla których tworzony jest finalny produkt. W powyższym przykładzie *reguła powielania* generującą znaczniki stosowała interpolację liniową pozycji nowego znacznika na podstawie dwóch wcześniejszych. Kolejne rozwiązania można aplikować poprzez dodawanie następnych *funkcji konwerterów* do puli wykorzystywanych przez reguły generujace znaczniki.

Warto zauważyć, że produkt transformacji jest materiałem gotowym do wykorzystania jako podstawa do tworzenia różnorodnych prezentacji trójwymiarowych.

Powyżej przedstawiowo sposób wykorzystania opracowanych rozwiązań w konkretnym zastosowaniu inżynieryjnym. Ważne jednak jest samo wykazanie możliwości tworzenia gramatyk K-GK rozbudowujących strukturę *kształtu graficznego* zarówno w ramach realizacji procedur rekursywnych jak i sekwencyjnych. To kluczowy element ścieżki dowodowej prowadzącej do wykazania przydatkości modelu K-GK. Opisane rozwiązania polegające na przygotowaniu przykładowych gramatyk umożliwiających realizację opisanych procesów potwierdzają celowość przyjętych w rozdziale piątym założeń. Powielanie proponowanych wzorców w zastosowaniach inżynieryjnych pozwoli na sprawne tworzenie rozbudowanych graficznych systemów generacyjnych opartych o model K-GK. Jest to elementem tezy głównej niniejszej pracy.

7.1.2 Wspomaganie modelowania transformacji dynamicznych

Kolejnym zagadnieniem jest potencjalna możliwość tworzenia gramatyk K-GK przygotowanych do prowadzenia ciągłej i dynamicznej rekonfiguracji *diagramu kształtu* po wcześniejszym utworzeniu struktury zawartych w nim obiektów. Wykazanie takiej przydatności umożliwi wspomaganie przetwarzania gramatykami dynamicznych scen trójwymiarowych, stanowiących arenę późniejszych gier 3D czy trójwymiarowych prezentacji. Testy prowadzone były w oparciu o zagadnienie kontrolowanego przemieszczania obiektów w trójwymiarowej przestrzeni projekcji z użyciem gramatyki K-GK. Innym zagadnieniem było tworzenie i modyfikoanie gramatykami trójwymiarowych brył wieloelementowych zbudowanych na bazie szkieletu bones.

Przemieszczanie obiektów mobilnych

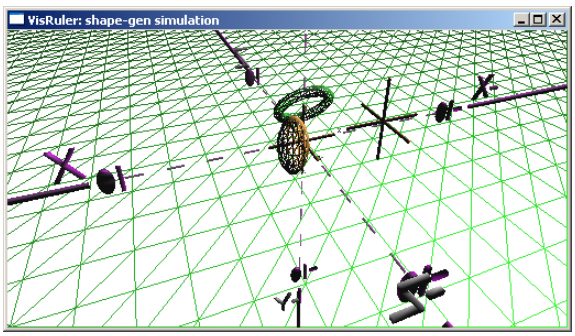
Po odpowiednim przy gotowaniu gramatyki K-GK testowy system został wykorzystany do projektowania rozwiązań wspomagających sterowanie obiektami, reprezentowanym przez *kształt graficzny* wraz z meta-danymi opisujących geometryczną lokalizację. Przetwarzanie generacyjne będzie tu prowadzone w czasie rzeczywistym. Pośród testów projektowanych narzędzi znajdą się tym samym także takie, które wykażą przydatność rozwiązania w procesach wspomagania decyzji dla prostych modułów sterujących ruchem obiektów mobilnych.

Metodyka przygotowania takiego rodzaju testu zakłada utrzymywanie obiektu sterowanego oraz jego środowiska w jednym zbiorze *kształtów graficznych* na *diagramie kształtów*. Obiekt sterowany nie będzie specjalnie wyróżniony, lecz to on będzie podlegał transformacji. Takie zachowanie można uzyskać dzięki:

- wyrażeniu stanu obiektu jako hybrydy jego aktualnej *klasy kształtu* i geometrycznej lokalizacji,
- zastosowaniu *reguł substytucji* jako środka do prowadzenia ciągłych modyfikacji tego stanu.

Zatem oprócz zmieniającej się lokalizacji w przestrzeni Euklidesa obiekt będzie także posiadał stan utożsamiony z *klasą kształtu*, jaką posiada. Tranzycje pomiędzy stanami będą wyrażone za pomocą *reguł substytucji*. Obiekt przechodzący w inny stan będzie *kształtem graficznym,* zmieniającym swoją *klasę kształtu*.

Na potrzeby eksperymentu przygotowano specjalny wariant środowiska, umożliwiającego trójwymiarowe śledzenie konkretnego *kształtu graficznego* jako punktu odniesienia (Rysunek 7.13).



Rysunek 7.13: Środowisko VisRuler - śledzenie sekwencji operacji modyfikujących *kształt funkcjami konwerterów* na ścieżce czasowej (źródło własne).

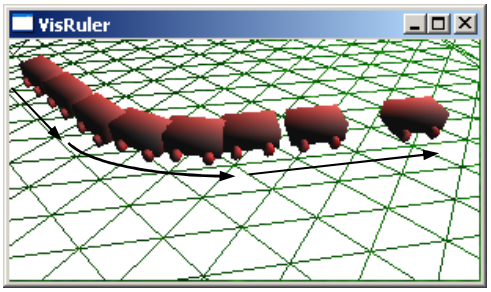
Pierwszy test polegał na zakodowaniu operacji sekwencji ruchu dla każdego z napotkanych w *diagramie kształtów* obiektów. Testowa sekwencja *reguł substytucji* będzie przemieszczała wytypowany *kształt graficzny* krokowo, w toku realizowania kolejnych iteracji przetwarzania. Późniejsze warianty eksperymentu będą zakładały uzależnienie przeprowadzania obiektu w nowy stan od obecności komponentów środowiska – wymuszone za pośrednictwem *funkcji walidatora* w r*egułach substytucji.*

W celu uporządkowania nomenklatury w tym podrozdziale *kształt graficzny* geometrycznie przemieszczany w konsekwencji prowadzenia testów będziemy identyfikowali pod nazwą: *obiekt mobilny*.

Konstrukcja pierwszej gramatyki K-GK zakłada prowadzenie krokowej modyfikacji meta-danych obiektu mobilnego z użyciem *funkcji konwertera reguł sybstytucji* w taki sposób, aby reprezentacja graficzna *kształtu* była renderowana w kolejnych lokalizacjach trójwymiarowej przestrzeni projekcji odpowiadając sekwencji ruchu tego obiektu. W przypadku trywialnym – tworzymy wiedzę o ruchu pojazdu z rozbiciem na kroki elementarne przy użyciu modelu klas. *Reguły substytucji* przeprowadzają *kształt* z *diagramu kształtów* z jednej *klasy* do kolejnej według schematu (zapis uproszczony, *reguł substytucji* byłoby 7):

KlasaRuch1 → KlasaRuch2 → KlasaSkretLewo1 → KlasaSkretLewo2 →
→ KlasaSkretLewo3 → KlasaRuch3 → KlasaRuch4 → KlasaRuchSzybki1.

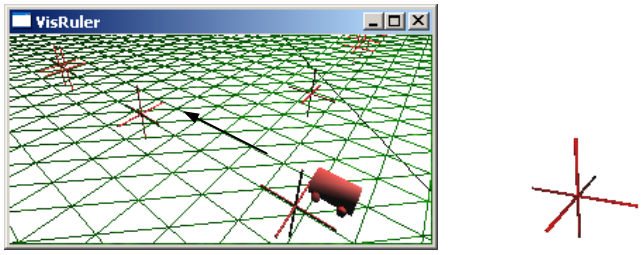
Diagram kształtów zawiera jeden *kształt* początkowy klasy KlasaRuch1. Każda z *reguł substytucji* posiada w LHS i RHS po jednym kształcie, *funkcje konwerterów* dokonują korekt meta-danych zawierających ułożenie w trójwymiarowej przestrzeni projekcji. Po zakodowaniu reguł i uruchomieniu animacji otrzymujemy kolejne pozycje obiektu mobilnego uzyskane po kolejnych iteracjach (Rysunek 7.14).



Rysunek 7.14: Sekwencja testowa przemieszczeń obiektu mobilnego w konsekwencji wymiany *klas kształtu* i realizacji *funkcji konwerterów* (źródło własne).

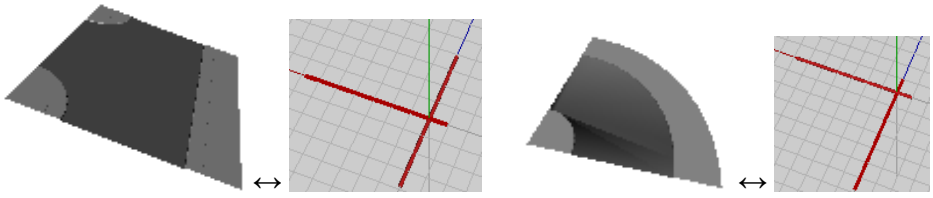
Angażowanie *modelu klas kształtów* do wyrażenia na nim każdego cyklu ruchu obiektu nie budzi jednak nadziei na wierne odwzorowanie choćby pojedynczej bardziej skomplikowanych sekwencji ruchu i ma małą wartość użytkową. Pośród wymagań wobec takiego rozwiązania stanie na pewno możliwość utrwalania sekwencji stanów, w jakie sterowane obiekty mogą przechodzić. Stworzenie aparatu decyzyjnego, umożliwiającego wyrokowanie o przeprowadzaniu obiektów pomiędzy ich stanami to kolejne zagadnienie. To

jest osiągalne jedynie przy posiadaniu możliwości interpretowania dodatkowej informacji pochodzącej ze środowiska obiektu mobilnego, na podstawie której będą podejmowane decyzje. Takiej informacji mogą dostarczyć meta-dane *kształtów graficznych*. Obiekty umieszczone w przestrzeni 3D obok obiektu mobilnego - a tym samym różnorodne *kształty graficzne* umieszczone w *diagramie kształtów* będą dostarczały informacji kontrolnej. Jest ona kontekstem, w którym modyfikowany będzie stan obiektu mobilnego. Kontekst jest uwzględniany w przypadku istnienia dodatkowych *kształtów* (znaczników), zlokalizowanych w sąsiedztwie obiektu mobilnego, a tym samym – w LHS reguł. Metrykę dopasowania zależną od tej odległości wyraża *funkcja walidatora*. Jeśli obiekt „zbliży się" do określonego znacznika (*kształtu* w *diagramie kształtów*) zostanie dopasowana *reguła substytucji*, zmieniająca jego *klasę kształtu*. Wówczas zmienia on stan i zaczyna on pokonywać kolejny etap swojej trasy (Rysunek 7.15).



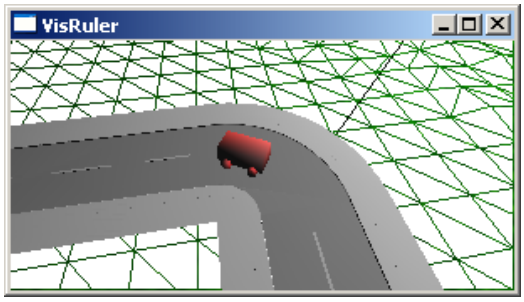
Rysunek 7.15: Trasa obiektu opatrzona znacznikami. Obiekt podąża ścieżką wytyczoną przez znaczniki (źródło własne).

Nic nie stoi na przeszkodzie, aby klasy znaczników także zróżnicować - różnicując w konsekwencji także zachowanie obiektu zbliżającego się do nich. Ponadto klasy tych *kształtów* mogą posiadać własne reprezentacje graficzne – uzupełniające wizualizację 3D (Rysunek 7.16). Nie będą wtedy jedynie *kształtami klas abstrakcyjnych*.



Rysunek 7.16. Przykładowe reprezentacje graficzne *kształtów graficznych* będących znacznikami (źródło własne).

Dzięki takiemu rozwiązaniu *kształt graficzny* pełni **jednocześnie** funkcję kontrolną i prezentacyjną (Rysunek 7.17).



Rysunek 7.17: Trasa obiektu opatrzona znacznikami posiadającymi reprezentację graficzną wykorzystana w procesie sterowania obiektem (źródło własne).

Uruchomienie dopasowanej *reguły substytucji* powoduje wykonywanie operacji:

- geometrycznego przemieszczenia reprezentacji graficznej obiektu mobilnego poprzez korektę treści meta-danych *kształtu graficznego*,
- zmiany *klasy kształtu* obiektu mobinego na inną.

Baza wiedzy, służąca do testowania mechanizmu przemieszczania obiektu po ścieżce prezentowanej na ryzunku 7.17 jest konstruowana z założeniem istnienia znaczników: drogi prostej, znacznika zakrętu oraz znacznika końca drogi. *Klasa kształtu* sterowanego obiektu mobilnego będzie się zmieniała, gdy znajdzie się on w sąsiedztwie kolejnego elementu ścieżki (znacznika). Przykładowo: *klasa kształtu* pojazdJadacyProsto zostanie zastąpiona przez pojazdSkrecajacyWLewo. Opuszczenie zakrętu – czyli dotarcie do kolejnego znacznika – spowoduje operację odwrotną. Dotarcie do znacznika końca drogi spowodować powinno przejście obiektu mobilnego w stan terminalny (cel został osiągnięty).

Uruchamianie *reguł substytucji* musi także powodować aktualizację lokalizacji sterowanego obiektu mobilnego. Ważne jest utrzymanie właściwej kolejności *reguł substytucji* na liście, a tym samym właściwej kolejności dopasowywania. Jako pierwsze rozpatrywane są reguły zmieniające kwalifikację *kształtu* do *klasy kaztałtu* lub doprowadzające *kształt* do stanu terminalnego, w drugiej – reguły aktualizujące jego geometryczną lokalizację. Przekwalifikowanie obiektu mobilnego do innej *klasy kształtu* będzie miało pierwszeństwo przed aktualizacją geometryczną pozycji. *Reguła substytucji* zmieniająca *klasę kształtu* może także zmieniać jego geometryczną lokalizację (powodując wykonanie dwóch czynności jednocześnie). Istnienie relacji geometrycznego sąsiedztwa *kształtów* jest sprawdzane przez *funkcję walidatora reguły substytucji*. Funkcja ta określa odległość od punktu znajdującego się

przed obiektem mobilnym zwykłą metryką Euklidesową. Wynik porównywany jest z wartością stałą, określoną w tej funkcji.

Problem interpretacji kształtu ścieżki sterującej trajektorią ruchu

We wcześniejszych rozważaniach zakładano, że reprezentacja graficzna *kształtów* stanowiących znaczniki na ścieżce obiektu mobilnego jest uproszczona. Składając się z serii wierzchołków umieszczonych w przestrzeni projekcji dostarczała ona łatwej do wykorzystania informacji o wymaganej trajektorii ruchu obiektu mobilnego. Gdy reprezentacja ta jest złożona, a obiekt znacznika posiada rozbudowaną reprezentację graficzną (jest np. modelem 3D) konieczne jest opracowanie rozwiązania wspomagającego jej interpretację. Reprezentacja ta może być bowiem źródłem dalszych ograniczeń ruchu obiektu mobilnego. Rozwiązaniem okazała się tu tymczasowa dekompozycja *kształtu graficznego* stanowiącego znacznik. Nowo powstałe obiekty kontrolne dostarczą informacji umożliwiającej prowadzenie bardziej dokładnych obliczeń *funkcją konwertera reguł substytucji*. Pozycje nowych znaczników będą obliczane specjalnie przygotowaną do tego *regułą powielania*. Tymczasowość dekompozycji spowoduje, iż dodatkowe *kształty* (zawsze utrudniające swoją obecnością dopasowanie reguł) będą kasowane jak tylko ich wykorzystanie nie będzie konieczne. Znacznik będzie wówczas powracał do postaci zagregowanej:

C_A = {Om (ObiektMobilny), abstract Z (ZnacznikZakretu), Zl (ZnacznikLiniiWZakrecie), Zz (ZnacznikZewnętrznyZakretu)}

D = {Om, Zz}

R = {

(…)

(1) Om + Zz + Zl :=> Om + Zz + Zl, w1 (Om, Zz, Zl), k1 (Om, Zz, Zl)

(2) Zz + Zl –=> , w2 (Zz, Zl), NULL

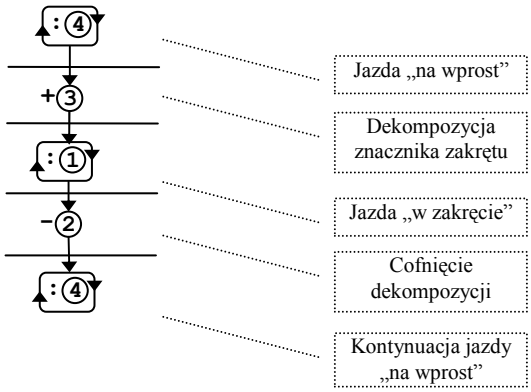
(3) Z + Om +=> Z + Om + Zz + Zl, w3 (Z, Om), k3 (Z, Om, Zz, Zl)

(4) Om :=> Om, NULL, k4 (Om)

}

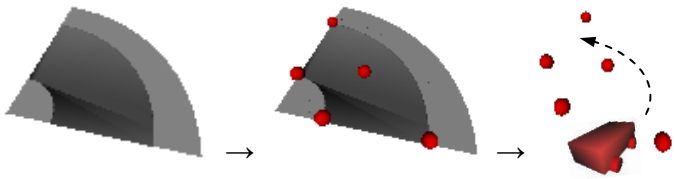
Reguła (1) (jak i ewentualne inne znajdujące się przed nią na liście *reguł substytucji*) będzie wykonywała korekty meta-danych obiektu mobilnego z użyciem znaczników zdekomponowanych. Gdy w *diagramie kształtów* nie będzie znaczników, lub w bieżącej iteracji nie zostały one zaakceptowane przez w1(Om, Zz, Zl) zostanie uruchomiona reguła (2) i

spowoduje usunięcie ewentualnych zdekomponowanych znaczników. Gdy znaczników nie było (Om dopiero zbliża się do znacznika) zostanie funkcją w3(Z, Om) sprawdzona reguła (3). Gdy znacznik Z jest dostatecznie blisko – zostanie zdekomponowany i Zz, Zl zostaną użyte w regule (1) w następnej iteracji. Naturalnie reguła (1) nie musi w LHS angażować wszystkich znaczników zdekomponowanych. *Kształt* Om będzie w rozwiązaniu reprezentował na przykład obiekt mobilny skręcający. Diagram K-GK dla takiej sytuacji został zaprezentowany na rysunku 7.18.



Rysunek 7.18: Diagram K-GK – dekompozycje znaczników ze ścieżki (źródło własne).

Niech przykładowo kształt zakrętu drogi posiada z góry znany promień i szerokość toru jazdy. Możliwe jest wtedy lepsze zapisanie treści *funkcji konwertera* uruchamianej gdy obiekt mobilny pokonuje taki zakręt. Gdy reprezentacja graficzna obiektów stanowiących ścieżkę jest skomplikowana możliwe będzie przeprowadzenie chwilowej dekompozycji tego *kształtu* na pod-kształty (Rysunek 7.19). *Funkcje konwartera* prowadzące obiekt mobilny w zakręcie będą się wtedy opierały na tych pod-kształtach.



Rysunek 7.19: Dekompozycja znacznika (źródło własne).

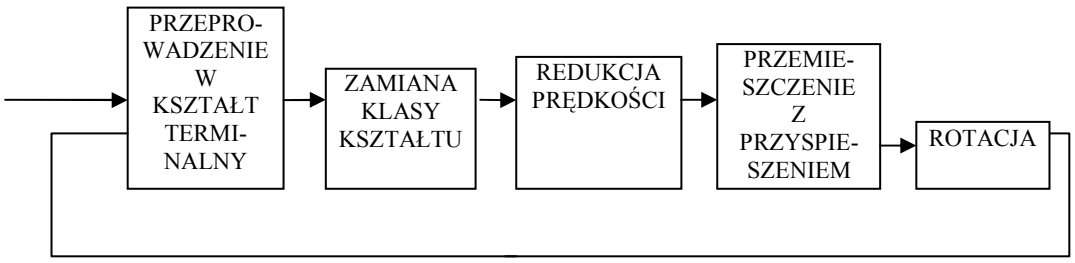
Warto zauważyć, że w przypadku braku możliwości zastosowania znaczników zdekomponowanych zostaną one automatycznie usunięte, a obiekt mobilny będzie przemieszczany poprzez wykonanie funkcji konwertera (4) zgodnie z rodzajem jego *klasy kształtu*.

Drogi alternatywne dla obiektu mobilnego

W przypadku rozwidleń dróg opatrzonych znacznikami obiekt sterowany musi zostać przeprowadzony w stan umożliwiający podążanie właściwą z nich. Wiadomo już, że stan ten będzie wymuszony przez chwilową zmianę *klasy kształtu* pojazdu z zachowaniem jego meta-danych. Temat algorytmicznego poszukiwania drogi na grafie połączeń, jaki można zbudować ze znaczników opisujących możliwe ścieżki jest ogólnie znany i nie będzie podlegał rozważaniom. Zakładamy zatem, że droga obiektu mobilnego jest już znana. Pozostaje do rozwiązania problem jej wyrażenia za pośrednictwem zbiorów gramatyki K-GK. Pośród kilku rozważnych rozwiązań najbardziej wartościowe okazało się dodanie kolejnych *kształtów* abstrakcyjnych (znaczników) wskazujących tą drogę i ustawionych przy rozgałęzieniach ścieżek. Są one odpowiednikami „znaków drogowych" przed skrzyżowaniem. Można naturalnie zrezygnować z abstrakcyjnej formy wyrażania tych znaczników i wizualizować je.

Problem sterowania prędkością obiektów

Z uwagi na otwarty format meta-danych możliwe jest dodanie do nich kolejnych atrybutów charakteryzujących dany *kształt*. Wszystkie są dostępne jedynie dla *funkcji konwerterów* i *funkcji walidatorów* w czasie ich wykonywania. Nie więc nie stoi na przeszkodzie, aby funkcje te operowały na kolejnych atrybutach charakteryzujących dalsze właściwości fizyczne obiektu (na przykład wektor prędkości, wektor momentu pędu czy prędkości obrotowej). Testując takie właściwości modelu przetwarzania zmodyfikowano *funkcję konwertera* reguły przemieszczającej obiekt „na wprost" tak, aby zwiększał on dodatkowo prędkość gdy ta jest mała. Dodano także *regułę substytucji*, która określała moment rozpoczęcia wytracania prędkości (na przykład) dwukrotnie szybciej, niż miało to miejsce w przypadku przyspieszania. Zapisana została bezpośrednio przed regułami „przemieszczającymi" obiekt mobilny (Rysunek 7.20).



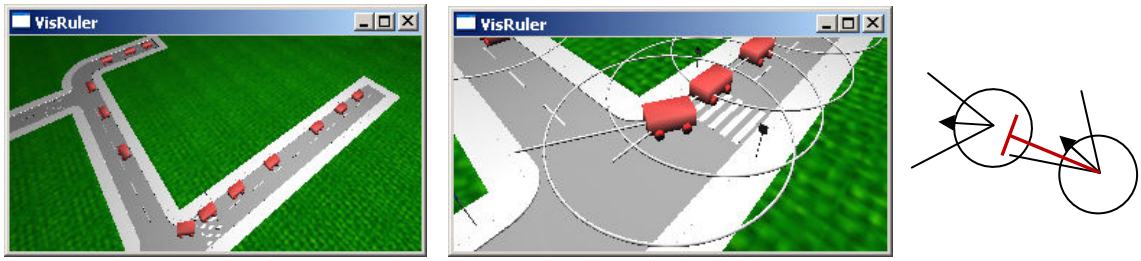
Rysunek 7.20: Kolejność *reguł substytucji* na liście (kolejność sprawdzania) (źródło własne).

Naturalne natrafienie na *regułę substytucji* naniesioną na powyższy rysunek nie musi oznaczać jej uruchomienia, więc każda z oznaczoych na rysunku 7.20 akcji może w danej iteracji nie mieć miejsca.

Zadanie postawione przy wcześniejszych rozważań polegało na geometrycznie wyrażonym przeprowadzeniu obiektu mobilnego do punktu docelowego po skomplikowanej ścieżce wyrażonej za pomocą znaczników. Gdy obiekt osiągnie cel – *kształt graficzny*, jakim jest reprezentowany staje się *kształtem terminalnym*. Teraz można podjąć próbę zdefiniowania gramatyki zarządzającej kolejkowaniem obiektów mobilnych, kontolując ich prędkości przemieszczania. Z uwagi na obecność innych obiektów mobilnych poruszających się po tej samej ścieżce konieczne było rozszerzenie algorytmu sterującego prędkością każdego z nich. Polegało ono na dodaniu *reguły substytucji*:

Om + Om +=> Om + Om, checkDistAndAngle (Om, Om), stop(Om, Om)

Funkcja validatora checkDistAndAngle(Om,Om) wykrywa obecność innego Om w przestrzeni zadanej wartościami kątowymi tylko wtedy, gdy przetwarzany *kształt* jeszcze się przemieszcza (nie stoi). Gdy reguła zostanie zakwalifikowana do uruchomienia, stop(Om) zatrzymuje pojazd (Rysunek 7.21).



Rysunek 7.21: Sterowanie prędkością obiektów mobilnych w przestrzeni projekcji (źródło własne).

Dalsze przykłady rozwijania zbioru *reguł substytucji* gramatyki K-GK w celu osiągnięcia kolenych, podonych do przedstawionych wyżej konstrukcji można mnożyć. Poruszone zagadnienie ukazało potencjalne możliwości wykorzystania opartego gramatykach K-GK systemu generacyjnego w przetwarzaniu ciągłym *kształtów graficznych*. Otwarta interpretacja meta-danych *kształtu graficznego* daje możliwości rozszerzania procedur obliczeniowych modyfikujących geometryczne usytuowanie *kształtu*. Z drugiej strony – posiadanie jakościowo uporządkowanego *diagramu kształtów* umożliwia szybkie przeszukiwanie jego treści regułami i określanie różnorodnych list obiektów wchodzących w ewentualną interakcję.

Zagadnienie przetwarzania kształtów graficznych w systemie bones

Do dalszych rozważań wytypowano jedno przykładowe zagadnienie wyspecjalizowane. To kolejne zagadnienie, w którym podjęto próbę zastosowania systemu utworzonego na bazie modelu K-GK. Dotyczy ono przetwarzania obiektów konstruowanych na bazie szkieletu bones. Rozwiązania tu zaangażowane stanowią hybrydę wcześniej przedstawionych, rozwiązując zarówno problem wygenerowania szkieletu bones jak i sterowania jego komponentami. Szkielet bones jest kompozycją wierzchołków i krawędzi drzewa (czasem ogólniej: grafu), gdzie krawędź drzewa stanowi element geometrycznej bryły modelu [Zho99]. Wierzchołek natomiast jest punktem w przestrzeni projekcji, wyrażającym zadaną dodatkowo relacją geometryczne powiązania między krawędziami. W praktyce ta relacja sprowadza się do wprowadzenia ograniczeń wzajemnego geometrycznego usytuowania krawędzi. Pośród tych ograniczeń kluczową rolę ma ułożenie kątowe dwóch krawędzi, zadane przedziałem dozwolonych wartości. To ograniczenie w szkielecie bones wprowadza stopnie swobody ruchu poszczególnych ich elementów względem siebie [Che03]. Szkielet taki jest wykorzystywany w procesach modelowania kinematyki licznych kształtów trójwymiarowych.

Istnieje wiele systemów wpierających modelowanie bones [Bln10]. Sztandarowym przykładem efektu transformacji bones jest modelowanie obiektów reprezentujących wirtualnego humanoida. Postać krocząca, posiadająca dwie nogi, tyłów, dwie ręce i głowę to stały element wielu interaktywnych animacji 3D. Naśladownictwo jej zachowania doprowadziło do opracowania wielu koncepcji modelowania wyglądu i kinematyki obiektów graficznych w bones [Has05]. Zastosowanie *modelu* K-GK powinno być przydatne i w tym przypadku.

Eksperyment ma na celu zaproponowanie gramatyki K-GK kontrolującej kinematykę brył trójwymiarowych opisanych *diagramami kształtów* i opartych na systemie bones. Testowym szkieletem będzie model pięciopalczastej dłoni – z możliwością jej otwierania i zamykania. Przetwarzanie *reguł substytucji* powinno dostarczyć mechanizmów wspierających ruch komponentów *diagramu kształtów* w ramach szkieletu bones (struktury opisującej bones). Struktura ta, niezależnie od faktu czy posiada reprezentację graficzną czy nie, musi zostać zapisana w *diagramie kształtów* gramatyki. Konieczne tutaj są:

- wyspecyfikowanie zaangażowanych w szkielet bones *klas kształtów* i relacji pomiędzy nimi w celu wyrażenia logicznych powiązań konkretnych komponentów szkieletu (np. głowa humanoida, dłoń, nadgarstek itp.),

- określenie *reguł substytucji* przetwarzających szkielet, czyli przemieszczających jego elementy względem siebie w ramach dozwolonych przez powiązania logiczne stopni swobody,
- aplikacja rozwiązania uzależniającego proces przetwarzania elemetnów bones od środowiska, co dostarczy możliwości ciągłego sterowania szkieletem bones,
- przygotowanie systemu testowej wizualizacji.

Modele 3D, będące niepodzielnymi reprezentacjami graficznymi komponentów *diagramu kształtów* w tym zagadnieniu zostały przygotowane w formie trójwymiatowych siatek wielokątów. Będą potrzebne przy bieżącej wizualizacji wyników. Gramatyki K-GK posłużą zarówno do wygenerowania *diagramu kształtów* reprezentującego dłoń, jak i do późniejszego sterowania komponentami tego diagramu. Generowanie szkieletu testowego sprowadzi się do podjęcia środków bardzo podobnych do używanych już w rozdziale 7.1.1. Znacznik P będzie przekształcany po powieleniu w przyszłe „palce" (P_1) regułą (2). Następnie regułą (1) w element „nadgarstka" (z_0):

(1) $P + P_1 + P_1 + P_1 + P_1 :=> z_0 + P_1 + P_1 + P_1 + P_1 + P_1,$ TRUE , NULL)

(2) $P +=> P + P_1 + P_1 + P_1 + P_1 + P_1$, TRUE,
kon2 (IN P, OUT P_1, OUT P_1, OUT P_1, OUT P_1, OUT P_1)

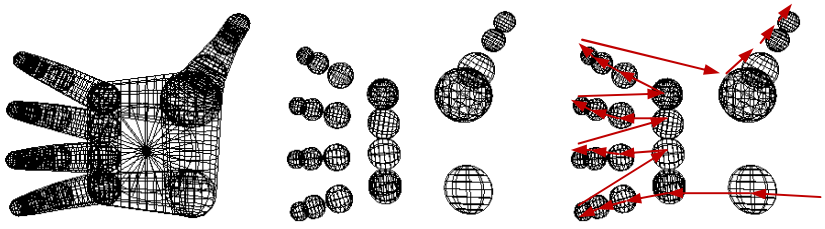
Później będzie miało miejsce genegowanie dalszych segmentów owych „palców" z równoczesnym tworzeniem znaczników $z_1..z_4$ (Rysunek 7.22). Reguły (3), (4), (5), (6) zostaną użyte po 5 razy (raz dla każdego ze znaczników):

(3) $P_1 +=> z_1 + P_2,$ NULL, kon3 (IN P_1, OUT z_1,OUT P_2)

(4) $P_2 +=> z_2 + P_3,$ NULL, kon4 (IN P_2, OUT z_2,OUT P_3)

(5) $P_3 +=> z_3 + P_4,$ NULL, kon5 (IN P_3, OUT z_3,OUT P_4)

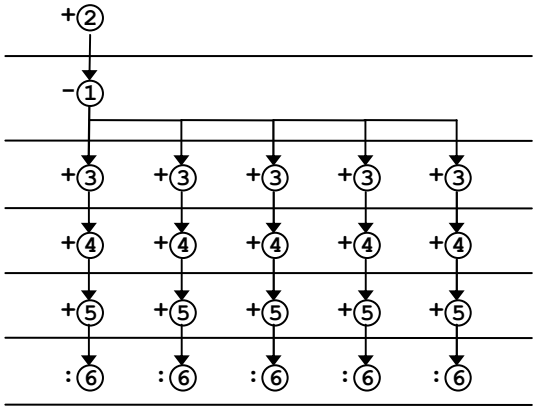
(6) $P_4 :=> z_4,$ NULL, kon6 (IN P_4, OUT z_4)



Rysunek 7.22: Generowanie obiektu dłoni gramatyką (po lewej), *kształty* kontrolne – znaczniki (pośrodku), procedura generowania znaczników (po prawej) (źródło własne).

Kształtu tymczasowego $P_1..P_4$ nie trzeba walidować, gdyż jest przekształcany dalej bezkontekstowo, a jedynym źródłem jego powstawania jest reguła (2).

Diagram K-GK dla procesu konstruowania układu znaczników $z_1..z_4$ został przedstawiony na rysunku 7.23.



Rysunek 7.23: Diagram K-GK – generowanie obiektu dłoni (źródło własne).

Druga gramatyka powinna modyfikować szkielet złożony ze znaczników $z_1..z_4$. Przedmiotem modyfikacji będą meta-dane znaczników, a konkretnie wartość kątowa zapisanej wektorowo rotacji znacznika. Wartość ta była już modyfikowana przez funkcje kon3()..kon6 w procesie tworzenia samych *kształów*. Przy pomocy wartości kątowych będzie możliwe sprawne rysowanie obiektu dłoni – wykonując rotacje (o wartość kątową ze znacznika) i translacje bieżącej pozycji rysowania w bibliotece graficznej. Celem tych operacji jest renderowanie kolejnego komponentu *diagramu kształtów* (czyli *kształtu graficznego* reprezentującego kolejny segment dłoni) w bieżącej pozycji rysowania.

Sterowanie obiektem bones wymaga pobrania ze środowiska rozkazu wykonania transformacji. Można to zrobić przez ciągłe aplikowanie do *diagramu sterujacego* jeszcze jednego znacznika (między poszczególymi iteracjami). Jego meta-dane określą parametry „rozkazu" do wykonania w następnej iteracji. Do aplikowania takiego znacznika można wykorzystać opisywany już w rozdzale piątym mechanizm generalizacji *kształtów*. Znacznik jest tymczasowy i zostanie usunięty w czasie wykonywania iteracji. *Reguły substytucji* są tak sformułowane, iż konwersja *kształtów* zachodzi gdy ten znacznik tymczasowy istnieje w *diagramie kształtów*. Funkcje konwerterów dokonują przekształcania innych znaczników na *diagramie kształtów*:

(1) $Z_x + Z_1 :=> Z_x + Z_1$, w1 $(Z_x + Z_1)$, k1 $(Z_x + Z_1)$

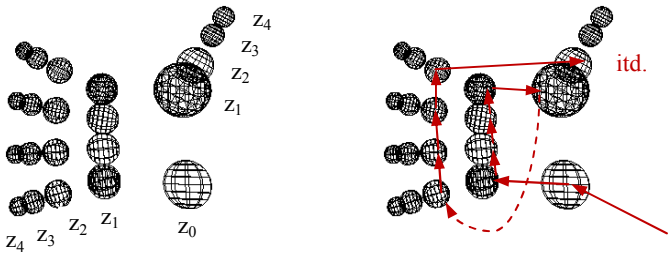
(2) $Z_x + Z_2 :=> Z_x + Z_2$, w2 $(Z_x + Z_1)$, k2 $(Z_x + Z_2)$

(3) $Z_x + Z_3 :=> Z_x + Z_3$, w3 $(Z_x + Z_1)$, k3 $(Z_x + Z_4)$

(4) $Z_x + Z_4 :=> Z_x + Z_4$, w4 $(Z_x + Z_1)$, k4 $(Z_x + Z_4)$

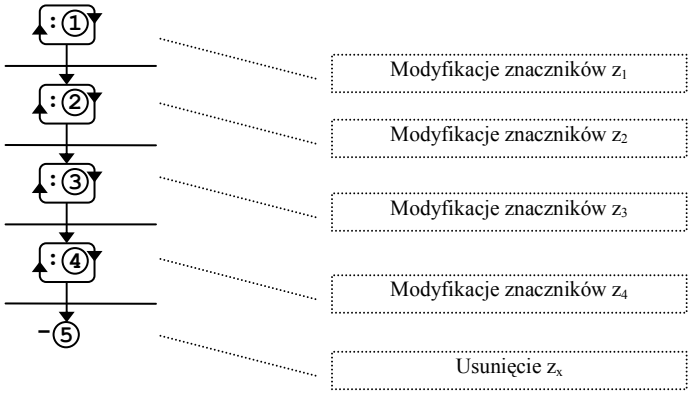
(5) Z_x –=> ,NULL, NULL

Funkcje validatorów w1..w4 wprowadzają dodatkowo ograniczenia kątowe dla wszystkich połączeń bones. Kolejność aplikowania zmian w meta-danych znaczników (korekta „rotacji") pokazuje rysunek 7.24:



Rysunek 7.24: Kolejność aplikowania zmian parametrów geometrycznych znacznika modyfikującego i kolejność aktualizacji znaczników *regułami substytucji* (źródło własne).

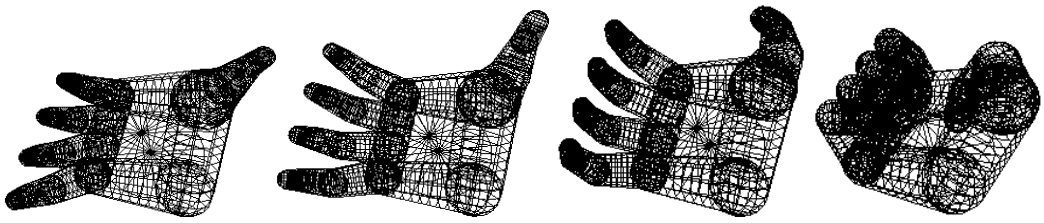
Diagram K-GK dla modyfikacji znaczników dłoni będzie dość prosty (Rysunek 7.25):



Rysunek 7.25: Diagram K-GK dla procesu aplikacji zmian właściwości geometrycznych lokalizacji znaczników z_1 .. z_4 (źródło własne).

Uzyskanie końcowego efektu graficznego wymaga jeszcze wygenerowania wypełnienia przestrzeni projekcji pomiędzy znacznikami, dodając tam *kształty graficzne* jednej i tej samej *klasy*. W czasie generowania gramatyką *kształty* te muszą ulec „przeskalowaniu" ze współczynnikiem zapisanym w ich meta-danych, aby pasowały graficznie do poszczególnych segmentów „palców" (rysunek 7.26). Ich reprezentacja graficzna to siatka stożka ściętego.

Wypełnienie samej dłoni reprezentowane jest wyjątkowo przez *kształt* jeszcze innej klasy, z reprezentajcą graficzną będącą prostopadłościanem:



Rysunek 7.26: Wyniki renderowania diagramu *kształtów* – szkielet bones (źródło własne).

Renderując powyższe wyniki wykazano możliwość wykorzystania gramatyk K-GK w roli wysokowydajnego mechanizmu umożliwiającego rozwijanie kinematyki bryły trójwymiarowej, której architektura oparta jest na systemie bones. Wybranie wyspecjalizowanego modelu (dłoń pięciopalczasta) nie zawęża tu dowodu przydatności, gdyż każdy szkielet bones można wyspecjalizować z grafu do postaci drzewa umieszczonego w trójwymiarowej przestrzeni projekcji. Przeprowadzony eksperyment polegał na sprawdzeniu możliwości welokrotnego iteracyjnego modyfikowania *diagramu kształów* gramatyką K-GK z jednoczesnym podawaniem do niej czynnika zewnętrznego sterującego transformacją. Zabieg ten, choć wykraczajacy nieco poza metodykę (*diagram kształtów* powinien być modyfikowany między iteracjami jedynie przez gramatykę), pozwoli prowadzonym przez gramatyki K-GK transformacjom na nabycie cech procesu interaktywnego.

7.2 Analiza własności poza funkcjonalnych systemu

Rozważania prowadzone w tym podrozdziale sprowadzać się będą głównie do oceny wydajności algorytmów przetwarzających *kształty graficzne*, zaimplementowanych w przykładowym systemie. Inne cechy poza funkcjonalne systemów opartych o model K-GK zostały już określone w rozdziale szóstym na etapie analizy ich architektury.

7.2.1 Wydajność algorytmu dopasowania reguł substytucji

Zilustrowanie wyników określających wydajność algorytmu poszukiwania *kształtów* w bazie wiedzy oraz dopasowywania *reguł substytucji* wymaga wprowadzenia do testowego komponentu przetwarzającego *reguły substytucji* systemu ewaluacji wyników. Precyzyjny czasu

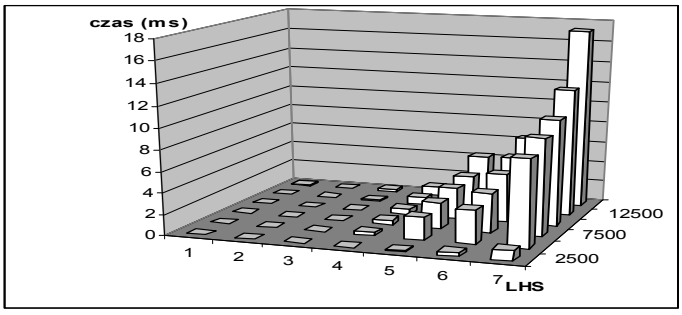
wykonania dopasowania jest możliwy po ingerencji w implementację komponentu przetwarzającego *reguły substytucji*. Do implementacji dodano „nadzorcę" wywołującego procedurę dopasowania *reguły substytucji* w trybie odseparowania. Nadzorca wykorzystuje specjalną bibliotekę umożliwiającą precyzyjny pomiar czasu wykonania podprocedury [Cod01]. Instancja klasy z tej biblioteki posiada timery, które można aktywować bezpośrednio przez wywołaniem funkcji realizującej badany czasowo proces. Pobranie wyniku następuje po powrocie z tej funkcji. Dokładność pomiaru to jedna mikrosekunda. Przy wygaszeniu innych procesów w systemie operacyjnym, oraz nadaniu procesowi badanemu priorytetu *realtime* wynik pomiaru jest bardzo precyzyjny. W ramach doświadczeń rejestrowano kolejno czas potrzebny do:

- poszukiwania nowego zestawu *kształtów graficznych* dla danej *reguły substytucji* - podawanego następnie do *funkcji walidatora*,
- sprawdzenia całej *reguły substytucji* przy założeniu, że wykonanie *funkcji walidatora* nie zajmuje czasu CPU i zawsze zwraca wartość *false* (czyli: 'szukaj dalej'),
- przeszukania całego zbioru *reguł substytucji*.

Problem wydajności algorytmu jest szczególnie istotny w przypadku prowadzenia wspomaganych sprzętowo przekształceń w czasie rzeczywistym. Wówczas kod stanowiący implementację algorytmu dopasowywania *reguł substytucji*, w przeciwieństwie do modułu zarządzającego reprezentacjami graficznymi, będzie nadal wykonywany jeszcze na (stanowiącym wąskie gardło) CPU. Procesor graficzny przejmie tylko przetwarzanie meta-danych *kształtów graficznych* oraz renderowanie siatek wielokątów, będących ich reprezentacjami graficznymi. Do prowadzenia pomiaru wydajności konieczne było przygotowanie rozległej bazy wiedzy o *kształtach graficznych* oraz *diagramu kształtów*, które skutecznie obciążyły by przetwarzający je komponent. Materiały te zostały sztucznie wygenerowane. Założono stopniowe powiększanie *diagramu kształtów* oraz liczby klas w kolejnych doświadczeniach. Przyjęto, iż klas jest 10-cio krotnie mniej niż *kształtów* w diagramie. Zatem przykładowo przy testowaniu *diagramu kształtów* zawierającego 2500 *kształtów* angażowane jest 250 rozpoznawalnych *klas kształtów*. W eksperymencie nie ma znaczenia treść logiczna bazy wiedzy lub *diagramu kształtów* gdyż nie następuje modyfikacja tego diagramu (uruchamianie *funkcji konwerterów* czy wprowadzanie zmian w liście *kształtów*

graficznych należących do *diagramu kształtów* zostało zdeaktywowane). Powiązania pomiędzy *kształtami graficznymi*, a ich *klasami kształtu* zostały wylosowane, więc statystycznie rzecz ujmując każda *klasa kształtu* stanowi wzorzec dla średnio 10 *kształtów graficznych* na diagramie. *Klasy kształtów* zostały wygenerowane z użyciem losowo komponowanych łańcuchów, stanowiących ich przyszłe nazwy. Sam model klas był płaski (nie wprowadzono żadnych powiązań pomiędzy *klasami kształtów*, świadczących istnieniu relacji *gen-spec* pomiedzy nimi). Warto zwrócić uwagę, że badania przeprowadzono dla bieżącej implementacji przykładowego systemu. Dokonano niej licznych czynności optymalizujących. Jedna z nich to wprowadzenie identyfikatorów *klas kształtów* będących referencjami do struktur w pamięci. Referencje te posortowano w indeks, nadając całkowito-liczbowe klucze wyszukiwania - co umożliwia dostęp z użyciem algorytmów szybkiego wyszukiwania. Tym samym nie jest konieczne przeszukiwanie słownika nazw *klas kształtów* w celu odnalezienia *kształtu* wytypowanego w LHS lub RHS reguły. Zatem – zawartość łańcuchów nazw klas nie wpłynie na wynik eksperymentów.

Poszukiwanie nowego zestawu *kształtów graficznych* realizowane jest przez algorytm opisany w rozdziale 5.4.10. Implementacja tego algorytmu będzie przedmiotem pierwszej serii testów. Dotyczyły one sprawdzenia jednej *reguły substytucji* przy wykluczeniu narzutów obliczeniowych na kontrolowanie wybranych *kształtów funkcjami walidatorów* (te operując na przestrzeni euklidesowej mogą być dowolnie skomplikowane) (Rysunek 7.27):



Rysunek 7.27: Czasy trwania (ms) dopasowania *reguły substytucji* zależnie od ilości *kształtów graficznych* na *diagramie kształtów* (odpowiednio: 2500, 5000, 7500, 10000, 12500, 15000 *kształtów graficznych*) - (PC CPU: Athlon 2.1 GHz) (źródło własne).

Kluczowym wnioskiem z pomiarów czasów jest wytyczna o technice formułowania *reguł substytucji*. *Reguły substytucji* zawierające więcej niż 4 czy 5 *kształtów graficznych* w LHS powinny być dekomponowane na kilka odpowiadających im reguł, gdy tylko jest to możliwe.

Możliwość ta występuje na przykład w sytuacjach, kiedy celem jest dokonanie dużej ilości homogenicznych *kształtów*. Przykładowo: przesunięcie o stały wektor wszystkich wielokrotności czwórek obiektów klasy C1 można zapisać dość rozlegle ujętą *regułą substytucji* o treści:

R_8 = { C1+C1+C1+C1+C1+C1+C1+C1 → C1+C1+C1+C1+C1+C1+C1+C1,

Validatepos8C1(),movC1(C1,C1,C1,C1,C1,C1,C1,C1)}

a następnie dodać:

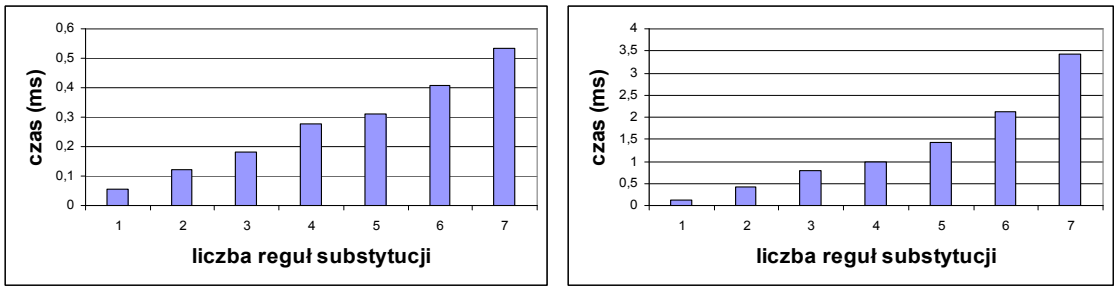
R_4 = {C1+C1+C1+C1 → C1+C1+C1+C1, Validatepos4C1(), movC1(C1,C1,C1,C1)}

licząc tym samym na podniesienie wydajności przetwarzania w związku z redukcją liczby aplikowanych reguł. Jednak korzystniejszym rozwiązaniem będzie usunięcie R_8 i przetwarzanie tylko regułą:

R_4 = { C1+C1+C1+C1 → C1+C1+C1+C1, Validatepos4C1(),movC1(C1,C1,C1,C1)}

pomimo powodowania uruchomienia tej drugiej większą ilość razy.

W przypadku pomiaru czasu trwania pełnego procesu podejmowania decyzji o uruchomieniu *reguły substytucji* wzięto pod uwagę *model kształtów* zawierający odpowiednio 5000 i 10000 *kształtów* dopasowywanych. Dla każdego z nich dopasowywano (z natychmiastowym odrzuceniem przez *funkcję walidatora*) kolejne *reguły substytucji*. Reguły te wygenerowano, losując zarówno wielkość LHS (z przedziału <1,4>) jak i rodzaj *klas kształtów* przypisywanych kolejnym *kształtom* w LHS (Rysunek 7.28)



Rysunek 7.28: Czas trwania (ms) procesu podejmowania decyzji o uruchomieniu *reguły substytucji* dla 5000 (po lewej) i 10000 (po prawej) *kształtów graficznych* zawartych w *diagramie kształtów* (PC CPU: Athlon 2.1 GHz) (źródło własne).

Przy niewielkich bazach wiedzy czasy potrzebne na obliczenie następnego kroku transformacji (wybranie *reguły substytucji* i uporządkowanej listy *kształtów graficznych*) są

bardzo krótkie (niemierzalne), co w przypadku projektowania systemów sterujących dla wielu homogenicznych obiektów pozwala stawiać tezę o dobrej skalowalności takich rozwiązań. Utwierdza to spektrum potencjalnych zastosowań metody w systemach wspierających generowanie interaktywnych scen trójwymiarowych w czasie rzeczywistym.

7.2.2 Renderowanie z wykorzystaniem reguł substytucji K-GK w warstwach wspomaganych sprzętowo

W obecnych czasach znaczna część obliczeń wspomagających przetwarzanie w renderowanej w czasie rzeczywistym grafice trójwymiarowej realizowana jest przy użyciu warstw akcelerowanych sprzętowo. Nie jest to tylko samo wyświetlanie gotowych siatek wielokątów, czy prowadzenie procesu ich teksturowania Zagadnienia akceleracji sprzętowej z uwagi na ciągły rozwój renderujących platform sprzętowych mogą także dotyczyć modyfikacji materiału graficznego zadanej wymaganiami funkcjonalnymi systemu. Jednym z rozwiązań technicznych umożliwiających taką modyfikację jest wspierana przez akceleratory 3D technologia shaderów zwana też w języku polskim po prostu „*cieniowaniem*". Obecnie rozważane zagadnienie będzie właśnie związane z próbą przeniesienia części funkcjonalności tworzonego systemu przetwarzającego *kształty* na graficzną platformę sprzętową. Niższe warstwy modelu K-GK, operując na geometrycznie określonej *reprezentacji graficznej* poszczególnych *kształtów* w *diagramie kształtów* stwarzają takie możliwości.

Programy cieniujące występują (zależnie od przeznaczenia) w kilku wariantach. Wśród nich można wyróżnić:

- *vertex shader*, który realizuje tzw. cieniowanie wierzchołkowe (uruchamiane raz dla każdego z przetwarzanych wierzchołków). Jego zadaniem jest transformacja położenia wierzchołka w wirtualnej przestrzeni projekcji. Ten proces określa się mianem wzmacniania lub osłabiania danych (*amplify or deamplify data*). Służy on zatem bezpośrednio do wspomagania renderowania *kształtu* na ekranie. Vertex shader może korygować współrzędne wierzchołka (także geometryczne atrybuty koloru, tekstury itp.). Możliwe jest więc teoretycznie wykorzystanie tej techniki do zakodowania mikroprogramów seryjnie przetwarzających wierzchołki w przestrzeni trójwymiarowej - jako procedur sterowanych gramatykami K-GK,

- *geometry shader*, który realizuje tzw. cieniowanie geometryczne. Procedury cieniowania geometrycznego są dużo bardziej rozbudowane. Umożliwiają operowanie na wielu wierzchołkach, w tym także ich dodawanie i usuwanie. W praktyce często wykorzystywane są do używane do proceduralnego tworzenia obiektów geometrycznych lub dodawania objętościowych detali do przetwarzanych siatek wierzchołków. Podobnie jak procedury cieniowania wierzchołkowego są obecnie implementowane dla GPU kart graficznych.

W uzupełnieniu do dwóch opisanych wyżej metod należy także wspomnieć o technologiach cieniowania pikseli (*pixel shaders*). Operują one wyłącznie na pojedynczych pikselach ekranowych urządzenia renderujacego. Manipulują kolorem piksela, korygując tą właściwość po pobraniu piksela z tzw. rasteryzatora.

W ujęciu generalnym technologia *vertex shaders* i *geometry shaders* zakłada wykorzystanie zestawów gotowych mikrokodów, które umieszczone w pamięci kodu akceleratora będą wykonywane na zlecenie głównego programu sterującego, przetwarzając dane wytypowane także przez ten program. Transformacje, możliwe do zakodowania nie mogą być obecnie zbyt skomplikowane (ograniczenia sprzętowe). Jednak już teraz dysponując siatką wielokątów opisaną za pomocą zbioru wierzchołków, jesteśmy w stanie dokonać znacznych modyfikacji. Spodziewany jest dalszy rozwój technologii *geometry shaders* i *vertex shaders*, który z pewnością umożliwi zwiększenie wymagań wydajnościowych wobec pojedynczych transformacji „cieniujących”.

Zaletą stosowania *vertex shaders i geometry shaders* jest fakt, że technologia ta pozwala na manipulowanie komponentami siatek 3D na poziomie tzw. *prymitywów*, czyli pojedynczych *kształtów* w nich zawartych. Z drugiej strony zamiast prowadzenia obliczeń na indywidualnych wierzchołach, istnieje możliwość działania także na nieco bardziej skomplikowanej strukturze. *Geometry shader* może przykładowo angażować do obliczeń wiedzę o strukturze siatki dostarczanej poprzez:

- pojedynczy wierzchołek (*single vertex*),
- krawędź (*line segment*) – co umożliwia manipulowanie krawędzią zdefiniowaną na przez dwa wierzchołki,
- trójkąt (*triangle*) – obróbka powierzchni definiowanych poprzez trzy wierzchołki.

Wykorzystanie omawianej technologii otwiera spektrum nowych możliwości w zakresie szybkiego przetwarzania *kształtów graficznych*. Możliwe stanie się implementowanie skomplikowanych procedur uruchamianych „w locie" przy przetwarzaniu potoku graficznego w GPU (*Graphics Processing Unit*).

Transformacje związane z *vertex shaders* dostępne są zarówno w przypadku użytkowania bibliotek graficznych Direct3D (wersja 9, 10 i 11) jak i OpenGL. Mikrokody są wykonywane podczas przetwarzania potoku graficznego, więc w przypadku transformacji wielo potokowych poszczególne procedury (shadery) mogą być realizowane współbieżnie. To kolejna zaleta technologii, poszerzająca spektrum możliwości wykorzystania. Przetwarzanie (niezależnie od kwestii współbieżności) w przypadku każdego z potoków następuje bezpośrednio przed realizacją programu *pixel shader* (koloryzacja pikseli). Procedury *geometry shaders* i *vertex shaders* produkują materiał przekazywany bezpośrednio do rasteryzatora i będą ostatecznie wpływały na kształt geometryczny prezentowanego materiału.

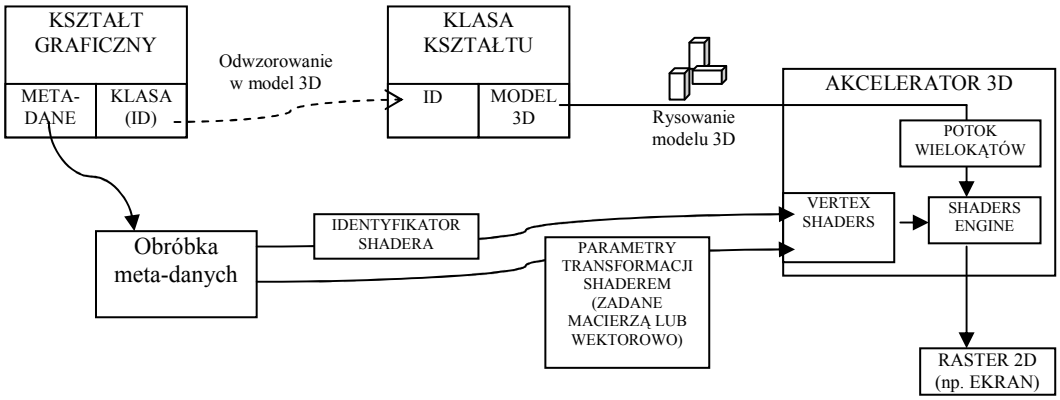
Wykazanie możliwości wykorzystania wspomaganych sprzętowo technologii przetwarzania grafiki do implementacji narzędzi stosujących model K-GK wymaga użycia konkretnego algorytmu sterującego przetwarzaniem 3D. Używany już w poprzednich eksperymentach system wizualizacji po wprowadzeniu pewnych modyfikacji nadal będzie tu przydatny. Tym razem umożliwi on sprawdzenie kolejnej teoretycznej możliwości zastosowania systemów opartych o gramatyki K-GK. W poprzednich akapitach omówiono możliwości przeniesienia implementacji gramatyk kształtu poszczególnych klas na platformy dostarczane przez sprzętowe akceleratory 3D.

Opisana technologia *vertex shaders* w praktyce umożliwia aktywowanie implantu kodu (vertex shadera), który zmodyfikuje pozycje wierzchołków bezpośrednio przed rasteryzacją do bufora 2D urządzenia renderującego [Lau04]. Wykonywany na GPU vertex shader gwarantuje wysoko wydajne przetwarzanie pozycji wierzchołków należących do siatek wielokątów w potoku graficznym akceleratora 3D. Zastosowanie Vertex shaders w silniku przetwarzającym *diagram kształtów* też powinno być możliwe. Będzie polegać na [Guh10]:

- przygotowaniu zestawu programów cieniujących (vertex shaderów),
- zapisaniu kodu ładującego i kompilującego Shadery w programie głównym,
- podaniu parametrów przetwarzania geometrii do vertex shadera,

- podania ewentualnej macierzy projekcji [Shr04] kamery do vertex shadera (shader operuje na bezwzględnym układzie odniesienia w przestrzeni euklidesowej, gdyż ewentualne macierze projekcji kamery zostały przetworzone przed utworzeniem potoku wielokątów w akceleratorze 3D, dlatego w kodzie shadera konieczne jest dokonanie konwersji wierzchołków do projekcji z uwzględnieniem pozycji kamery),
- aktywowania vertex shadera,
- narysowania *reprezentacji graficznej* danego *kształtu graficznego* **bez wykonywania** obliczeń związanych z interpretacją jego meta-danych.

Pozycjonowanie i inne konwersje *reprezentacji graficznej kształtu* na bazie meta-danych będą prowadzone właśnie przez vertex shader, dlatego w rozwiązaniu nie wykonujemy żadnych obliczeń modyfikujących siatkę wielokątów reprezentującą *kształt graficzny*. Vertex shader automatycznie przeliczy pozycje wierzchołków z potoku wielokątów, więc gdy jest aktywny wystarczy podać materiał do renderowania wywołując standardowe funkcje rysujące biblioteki graficznej 3D (w przypadku opracowania: OpenGL). Zasadę funkcjonowania testowego rozwiązania prezentuje rysunek 7.29:



Rysunek 7.29: Schemat procesu aplikowania vertex shadera przy interpretacji meta-danych *kształtów graficznych* w renderowaniu tego *kształtu* (źródło własne).

Tym samym funkcjonalność komponentu przetwarzającego meta-dane (te same, które są przedmiotem przetwarzania w *funkcjach walidatorów* i *funkcjach konwerterów reguł substytucji*) w celu renderowania *diagramu kształtów* przenosimy w dużej części na platformę

wspomaganą sprzętowo. Moduł zarządzający reprezentacją graficzną *klas kształtów* nie wymaga zmian.

Przygotowanie vertex shaderów

Do zakodowania shaderów zastosowano język Cg (C for Graphics) [Cgt10] oraz API CG firmy NVidia. Kod pisany w języku Cg to zestaw funkcji dokonujących konwersji wierzchołków w kilku wariantach zgodnie z treścią meta-danych dla każdego z *kształtów*. Programy shaderów dokonują przesunięć pozycji poszczególnych wierzchołków siatki wielokątów z potoku. Pobierają parametry dokonywanych transformacji z programu głównego. Z uwagi na konieczność przeniesienia wartości tych parametrów pomiędzy platformami (mamy tu program główny wykonywany na CPU oraz vertex shader realizowany przez GPU) został opracowany pomost standaryzujący format parametrów, których przekazanie jest dopuszczalne. Standard ten ewoluuje wraz ze zwiększającymi się możliwościami technicznymi akceleratorów 3D. W jego konsekwencji definiowane są tzw. profile vertex shaders [Fos03], określające pulę możliwych do przekazania i odebrania parametrów. W przypadku Cg pojedyncze parametry przenoszą wartości liczbowe w macierzy 4x4 (wartości typu float4x4), wektor wartości liczbowych (wartości typu float4) lub pojedynczy float [Nvi07]. Wybranie profilu określa naturę parametrów wejściowych i wyjściowych dla vertex shadera. Konieczne jest przesłanie macierzy lokalizacji *kształtu graficznego* z meta-danych oraz wektorów. Zostały tu wytypowane i spełniały wymagania dwa podstawowe profile Cg: *vs_1_1* lub *vs_2_0* [Nvi07]. Profile te umożliwiają zadeklarowanie głównego programu vertex shadera, który pobierze z profilu w/w parametry. Ograniczenia profili dyktowane są możliwościami sprzętu, więc zanikają.

Na potrzeby doświadczenia sam zbiór *vertex shaderów* konwertujących ponownie uproszczono do wykonujących podstawowe operacje geometryczne na wierzchołkach (rotacja, przesunięcie, skalowanie). Niewielki problem sprawiła jedynie implementacja rotacji, która teraz musi być prowadzona względem geometrycznego środka modelu 3D reprezentującego *kształt*. (przyjęto, że jest to pochodząca z meta-danych pozycja *kształtu graficznego*). W wariancie wektorowym vertex shadera obracającego model 3D – meta-dane dostarczają dla programu dwa dodatkowe parametry (zakodowane jako wektory): pozycję i rotację. Implementacja postaci przetwarzającej macierze sprowadza się jedynie do przeniesienia procedury mnożenia macierzy na inną platformę.

Modyfikacja komponentu zarządzającego projekcją *kształtów graficznych*

Konieczne było tu przebudowanie poprzednio stosowanej procedury wywołującej funkcje rysujące silnika 3D, których wywołania:

- definiowały lokalizację bieżącego punktu rysowania w wirtualnej przestrzeni projekcji 3D,
- powodowały rysowanie modelu 3D.

Wywoływanie rysowania ma nadal miejsce w nie zmienionym trybie, jednak pierwsza z czynności będzie teraz rozbita na:

- określenie vertex shadera wykonującego operację konwersji (na podstawie meta-danych),
- przygotowanie na podstawie meta-danych parametrów vertex shadera i aktywowanie go.

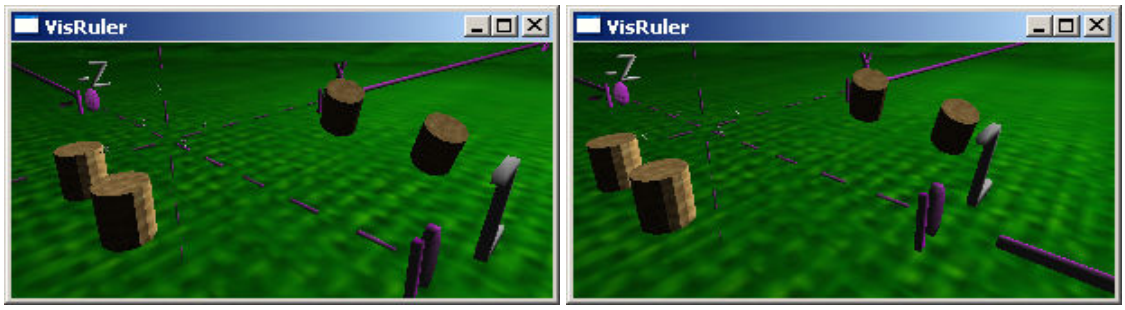
Rezultat modyfikacji

W doświadczeniu postanowiono udowodnić, iż możliwe jest zaangażowanie tego rozwiązania także w przypadku modelu K-GK. Wielowarstwowy model przetwarzania K-GK nie nie musi być specjalnie modyfikowany. Komponenty przetwarzające *reprezentację graficzną* i meta-dane *kształtów* należy jedynie w inny sposób implementować.

Wynikiem jest wyrenderowana klatka animacji, będąca wizualizacją *diagramu kształtów* graficznych wspomaganą przez technologię vertex shaders. Stworzenie odpowiednich shaderów okazało się osiągalne dla wszystkich możliwych wartości meta-danych, jakimi komponenty operujące w stworzone na potrzeby pracy operowały. Przypomnieć należy, iż dotyczy to zarówno przetwarzania wektorowego 3D jak i przy użyciu 16-to elementowej macierzy transformacji. Stosowanie vertex shaderów nie wprowadza niepożądanych zaburzeń do procesu teksturowania. Silnik 3D autora ładuje tekstury z plików dedykowanych i umieszcza z buforze tekstur akceleratora 3D [Sum04]. Przemieszczenie wierzchołka procedurą vertex shadera nie może wpłynąć negatywnie na skutki działania procedury teksturowania obiektów graficznych. Koordynaty tekstur [Wri00] nie ulegają co prawda zmianie, ale rasteryzacja tekstur do bufora 2D bazuje na pozycjach wierzchołków wielokątów, które były modyfikowane vertex shaderem. W niektórych przypadkach implementacji vertex shaderów swobodne korekty pozycji

wierzchołków już po umieszczeniu ich w potoku wielokątów powodowały niepożądane efekty uboczne teksturowania (niedorysowania). W konsekwencji konieczne jest narzucenie ograniczeń w programach vertex shaderów (przez np. przeniesienie do zbioru wartości dyskretnych wymiarów kątowych rotacji i przesunięć wierzchołków) [Buc04]. To narzuciło by ograniczenia w swobodzie interpretowania meta-danych *kształtu graficznego* dla tej technologii. Po licznych eksperymentach udało się jednak zakodować treść vertex shaderów w sposób niwelujący problem.

Wyniki wdrożenia opisywanego rozwiązania są widoczne dopiero po uruchomieniu systemu prowadzącego renderowanie wyników w czasie rzeczywistym. Wykazanie poprawności funkcjonowania rozwiązania może nastąpić poprzez wizualne porównanie dużych serii klatek animacji tworzonych odpowiednio: bez użycia technologii vertex shaders, z zastosowaniem shaderów operujących na macierzach i z zastosowaniem tych kodowanych w wariancie operującym na wektorach pozycji (Rysunek 7.30).



Rysunek 7.30: Wyrenderowane klatki (kolejno od lewej): bez vertex shaders, vertex shaders (przetwarzanie meta-danych *kształtów graficznych* - różnice geometryczne powodowane są wyłącznie rozbieżnościami w manualnie sterowanym ustawieniu kamery (źródło własne).

Dynamiczna korekta pozycji wierzchołków przez program vertex shadera jest pośrednim efektem działania *funkcji konwertera* zawartej w *regułach substytucji*.

Przeprowadzono także doświadczenia umożliwiające wykazanie, iż kod samych *funkcji konwertera* jest także możliwy do przeniesienia na platformę sprzętową. Wąskim gardłem takiego dla systemu przetwarzania okazały się ograniczenia opisywanego wcześniej profilu vertex shadera (profilu Cg) dostarczanego przy modyfikacji każdego wierzchołka w potoku wielokątów. Ilość danych przekazywanych do *funkcji konwertera* i *funkcji walidatora* w *regule substytucji* może być dowolnie duża (brak limitu ilości *kształtów graficznych* LHS lub RHS). Ograniczenia profilu narzuciły tu limit ilości *kształtów graficznych*, jakie można przekazać do

wykonywanej na GPU *funkcji konwertera*. Pomyślnie (na bazie profilu vs_1_1) zaimplementowano funkcje przetwarzające 4 *kształty graficzne*. Udowodniono tym samym wstępnie możliwość przeniesienia funkcjonalności kolejnej warstwy modelu K-GK na platformę akcelerowaną sprzętowo. Bardziej zaawansowane profile vertex shaders, powstając wraz z kolejnymi generacjami sprzętu będą poszerzały możliwości szybkiego przetwarzania geometrii elementów *diagramu kształtów* opisanej ich meta-danymi. Warto zatem rozważać realizacje kolejnych kroków adaptujących graficzne systemy generacyjne do funkcjonowania nad tymi technologiami.

8 Wnioski i kierunki dalszych prac

W niniejszej pracy przedstawione zostało nowe podejście do problemu pół-formalnego przetwarzania trójwymiarowych obiektów graficznych. Zaproponowano metodologię rozwijania oprogramowania dla graficznych systemów generacyjnych prowadzących transformacje w oparciu o wykorzystanie reguł. W ramach metodologii zaproponowano utworzenie wielowarstwowego modelu przetwarzania, którego komponenty operując na różnych poziomach abstrakcji interpretowania materiału graficznego będą wspierały funkcjonalnie graficzny system generacyjny. Opracowanie optymalnego rozwiązania wymagało szeregu badań w dziedzinie przetwarzania graficznych obiektów trójwymiarowych metodami pół-formalnymi.

W rozdziale siódmym pokazano, iż spektrum zastosowań systemów opartych na zaprezentowanej metodologii jest szerokie. Obejmuje dziedziny, w których proces przetwarzający trójwymiarowe komponenty graficzne dąży do rozbudowania ich struktury na bazie informacji klasyfikującej. Nie operuje przy tym jedynie na wyizolowanej przestrzeni symboli, lecz na rzeczywistym materiale podlegającym późniejszej wizualizacji. Wykazano celowość tworzenia zintegrowanych systemów klasyfikujących komponenty graficzne oraz relacje zachodzące pomiędzy nimi. Możliwość rozwijania tych systemów w skali globalnej przedstawiono w rozdziale szóstym. Opisano tam przykład interfejsu wymieniającego ontologicznie uporządkowaną wiedzę o odpowiednikach znaczeniowych słów języków naturalnych - zastosowanego do opisywania baz wiedzy o komponentach graficznych. W rozdziale szóstym zaprojektowano także rozwiązania umożliwiające łatwą interpretację wiedzy mogącej służyć do budowy gramatyk o proponowanej architekturze. Opracowano bazujący na XML język opisujący taką wiedzę. Przedstawiono rozwiązania techniczne umożliwiające bardzo szerokie określenie przestrzeni funkcjonowania proponowanych gramatyk K-GK – od prowadzenia operacji na modelu abstrakcyjnym przez przetwarzanie cech geometrycznych różnorodnego materiału graficznego do kwestii operowania na konkretnych reprezentacjach graficznych takiego materiału, czyli renderowania jego wizualnego wyglądu. Rozwiązania opracowywane w oparciu o proponowaną metodykę będą mogły funkcjonować także jako kompleksowe systemy wizualizacji, renderujące przetwarzane materiały w czasie rzeczywistym.

Systemy tworzone w oparciu o przyjęte założenia będą także mogły spełniać istotne wymagania poza funkcjonalne, takie jak rozszerzalność funkcjonalności warstw i łatwość ponownego wykorzystania ich elementów. Gwarantuje to komponentowa architektura tych systemów. Wykazano, iż w przypadku pojawienia się zapotrzebowania na zastosowania wymagające trójwymiarowej wizualizacji w czasie rzeczywistym proponowany model K-GK pozwala na wytwarzanie systemów wysokowydajnych. Systemy te mogą być silnie integrowane z niskopoziomowymi graficznymi platformami renderującymi, które stosowane są powszechnie w dzisiejszych stacjach komputerowych. Umożliwi to znaczny wzrost wydajności procesów przetwarzających materiał graficzny.

Jak wskazano w rozdziale trzecim, zagadnienia związane z automatyzacją procesów przetwarzania *kształtów graficznych* zapisanych jako obiekty trójwymiarowe są niezmiernie rozległe. Dotychczasowe próby opracowania metodyk formalnego przetwarzania *kształtów graficznych* zakładały zawężanie spektrum zastosowań przekształceń do określonych dziedzin tych *kształtów*. Zaprezentowane rozwiązanie otworzy drogę do ominięcia tego ograniczenia, formułując nie tylko wiedzę o samych *kształtach*, lecz także wiedzę o procedurach ich przetwarzania.

Prawdziwość głównej teza pracy została wykazana. Postulowano w niej, iż możliwe jest dokonanie takiej przebudowy modelu klasycznych gramatyk kształtu, aby cały proces decyzyjny dotyczący stosowania ich reguł przebiegał w pełni deterministycznie, z uwzględnieniem rozbudowanej wiedzy opisującej daną dziedzinę projektową, oraz był równocześnie prowadzony bez konieczności ingerencji jakiegokolwiek czynnika ludzkiego. Dzięki odłączeniu abstrakcji *kształtu graficznego* od jego graficznej reprezentacji możliwe stało się uniezależnienie przetwarzania *kształtów graficznych* od konieczności posiadania gotowych transformacji analizujących skomplikowane reprezentacje graficzne tych *kształtów*. Nie jest też konieczne algorytmiczne poszukiwanie pod-kształtów przetwarzanego materiału graficznego, aby te mogły ulegać dyktowanej regułą gramatyki wymianie lub konwersji (jak to miało miejsce w przypadku klasycznych gramatyk kształtu). Podstawę do tworzenia gramatyk stanowi obecnie rozbudowana strukturalnie wiedza, zaś *diagramy sterujące* K-GK umożliwiają dostarczenie do procesu przetwarzającego wiedzy pozwalającej na całkowicie autonomiczne jego prowadzenie.

Projektowanie systemów wykorzystujących proponowany model będzie ułatwione, gdyż:

- system taki będzie posiadał warstwową architekturę, umożliwiającą tworzenie nowych implementacji odizolowanych komponentów bez ingerencji w komponenty warstw pozostałych,
- model K-GK będzie w naturalny sposób ułatwiał tworzenie komponentów realizujących wizualizację,
- możliwe będzie przeniesienie licznych operacji związanych z przetwarzaniem komponentów graficznych na platformy graficzne wspierane sprzętowo.

Opracowanie języka CSGL umożliwia wyrażenie wiedzy potrzebnej do skonstruowania *gramatyki* K-GK w formie spójnego materiału, którego struktura oparta jest na tekstowych znacznikach. Ułatwi to wymianę wiedzy pomiędzy różnymi systemami. Umożliwi także konstruowanie gramatyk K-GK w oparciu o wyniki przetwarzania produkowane przez inne takie gramatyki, gdyż wyniki te także można opisać przy użyciu języka CSGL. Uporządkowanie semantyczne *kształtów graficznych* jest utrzymane za sprawą istnienia modelu *klas kształtów*, wprowadzającego w konsekwencji także uporządkowanie jakościowe. Ponieważ *klasa kształtu* gramatyki K-GK identyfikowana jest za pomocą słowa języka naturalnego, możliwe stało się opracowanie pomostu pozwalającego na poszukiwanie odpowiedników znaczeniowych *klasy kształtu* w innych bazach wiedzy. Tym samym – możliwe stało się poszukiwanie wiedzy o transformacjach graficznych uzupełniającej wiedzę aktualnie rozważaną. Po utworzeniu interfejsu do ontologicznych baz odpowiedników znaczeniowych – konstruowana przestrzeń nazw nabrała wspomnianego już charakteru globalnego. Powyższe stwierdzenia były postulowane przez tezy poboczne pracy.

Zaproponowane w pracy podejście do tworzenia graficznych systemów generacyjnych może stanowić podstawę dla dalszych prac i badań w tej dziedzinie. Ciekawy kierunek rozwoju reprezentują prowadzone już przez autora badania mające na celu dalsze rozszerzenie modelu K-GK. Zakładają one rozbudowę o konstrukcję umożliwiającą współbieżne modyfikowanie *diagramu kształtów* przez wiele gramatyk K-GK. Gramatyki posiadałyby własne zbiory produkcji, stanowiące wiedzę o metodach przetwarzania. Operowały by na tym samym materiale (*diagramie kształtów*), ale w różnych przestrzeniach abstrakcyjnie określonych dzielonym modelem *klas kształtów*. Dzięki ontologicznemu poszukiwaniu odpowiedników *klas kształtów* można by wówczas łączyć te przestrzenie i umożliwiać kolaborację różnych gramatyk K-GK. Skutkowałoby to otwarciem możliwości przekazywania niedokończonych elementów

diagramu kształtów pomiędzy poszczególnymi procesami wytwórczymi (czyli różnymi gramatykami K-KG).

Innym kierunkiem rozwoju jest dalsze podnoszenie poziomu funkcjonalnego zintegrowania graficznych systemów generacyjnych ze wspomaganymi sprzętowo trójwymiarowymi graficznymi platformami renderującymi. Wykazano, iż pełne wsparcie sprzętowe procesu renderowania definiowanego modelem K-GK *diagramu kształtów* jest osiągalne. Podobnie osiągalna jest sprzętowo wspierana realizacja rachunku przekształceń geometrycznych *kształtów graficznych* w gramatyce K-GK. Zaangażowanie wspomaganych sprzętowo graficznych platform renderujących do generowania kompletnych scenariuszy interaktywnych prezentacji trójwymiarowych powinno dać w zastosowaniach inżynieryjnych bardzo satysfakcjonujące wyniki.

Literatura

[3ds10] *The 3D Studio Home Site*, http://www.the3dstudio.com, 2010.

[And76] Anderson J., *"Language, Memory and Thought"*, Erlbaum Associates, Hillsdale, NJ, 1976.

[And93] Anderson J., *"The Architecture of Cognition. Cambridge"*, Harvard University Press, 1983.

[Ban02] Banaszak G., Gajda W., *"Elementy algebry liniowej"*, Wydawnictwa Naukowo-Techniczne, 2002.

[Bis04] Bischoff S., Kobbelt L., *"Teaching meshes, subdivision and multiresolution techniques"* Computer-Aided Design 36(14), 1483–1500, 2004.

[Bln10] *Blender for Developers WebSite*, http://www.blender.org/development, 2010.

[Bol00] Boley H., Tablet S., *„Design Rationale of RuleML: A markup Language for semantic web rules"*, Semantic Web enabled Web Services, 2000.

[Bot97] Bottoni P., *"Computing with Shapes"*, Turku Centre of computer Science, Technical Report 136, 1997.

[Buc04] Buck I., Foley T., Horn D., *"Brook for GPU's: Stream computing on graphics hardware"*, Transactions on Graphics 23 Vol. 3, 777-786, 2004.

[Cgt10] *Cg Toolkit - GPU Shader Authoring Language*, http://developer.nvidia.com/object/cg_toolkit.html, 2010.

[Cha96] Chase S., *"Using logic to specify shapes and spatial relations in design grammars"*, Grammatical Design, Fourth International Conference on Artificial Intelligence in Design, Stanford University, 1996.

[Chd02] Chuderski A., *"Wykorzystanie metod sztucznej inteligencji w badaniach nad umysłem"*, Rozprawa doktorska, 2002.

[Che03] Chen J., *"Guide to graphics software tools"*, Springer, 94, 205, 2003.

[Cho06] ShangChing C., XiaoShan G., *"A Rule Based Geometry Theorem Prover Using Fullangles"*, Department of Computer Science, The Wichita State University, 2006.

[Cod01] *The Code Project Development Resource*, http://www.codeproject.com/KB/recipes/precisetimer.aspx, 2001.

[Cun94] Cuny J., *"Graph grammars and their application to computer science"*, 5th International Workshop, Springer, 442-451, 1994.

[Dob92] Dobryjanowicz E., *„Podstawy przetwarzania języka naturalnego: Wybrane metody analizy składniowej"*, Akademicka Oficyna Wydawnicza RM, 163-164, 1992.

[Doo78] Doo D., Sabin M., *"Analysis of the Behaviour of Recursive Division Surfaces"*, Computer Aided, Design, volume 10, 356–360, 1978.

[Dua00] Duarte J., *"Customizing mass housing: a discursive grammar for Siza's Malagueira houses"*, Massachusetts Institute of Technology, 2000.

[Dua02] Duarte J., *"Malagueira Grammar towards a tool for customizing Alvaro Siza's mass houses at Malagueira."*, PhD Thesis, MIT School of Architecture and Planning, 2002.

[Dyd78] Dydak J., Segal J., *„Lecture Notes in Mathematics: Shape Theory"*, Springer-Verlag, Berlin, 26-30, 1978.

[Fle87] Flemming U., *"The role of shape grammars in the analysis and creation of designs"*, YE Kalay (ed), Computability of design, John Wiley, New York, 245–272, 1987.

[Fol90] Foley J.D., *"Wprowadzenie do grafiki komputerowej"*, WNT Warszawa, 474, 475, 1990.

[Fos03] Fosner R., *"Real-Time Shader Programming"*, Elsevier Science, 190-204, 2003.

[Ger04] Gero J.*"Evaluation Of A 3d Shape Grammar Implementation"*, Design Computing and Cognition, Kluwer Academic Publishers, 357-376, 2004.

[Ger06] Gero J., Suschil S. Gero, *"Evolutionary Learning of Novel Grammars for Design Improvement"*, AIEDAM, 8(2), 83-94, 2006.

[Gip75] Gips J., *"Shape Grammars and Their Uses: Artificial Perception"*, Shape Generation and Computer Aesthetics'75, 1975.

[Gip99] Gips J., *„Computer Implementation of Shape Grammars"*, Workshop on Shape Computation, MIT, 1999.

[Gra07] Grabska E., *"Projektowanie wizualne wspomagane komputerem"*, Wydawnictwo EXIT, 2007.

[Gru93] Gruber T., *"Toward Principles for the Design of Ontologies Used for Knowledge Sharing"*, Technical Report KSL 93-04 , Knowledge Systems Laboratory, Stanford University, http://tomgruber.org/writing/onto-design.pdf, 1993.

[Gru93] Gruber T., *"A translation approach to portable ontology specifications"*, Knowledge Acquisition 5, Stanford, CA, 199-220, 1993.

[Grz05] Grzesiak-Kopeć K., *„Wizualne projektowanie modeli graficznych z wykorzystaniem elementów emergentnych"*, rozprawa doktorska, AGH, EAIiE, 2005.

[Guh10] Guha S., *"Computer Graphics Through Opengl: From Theory to Experiments"*, CRC Press, 740-743, 2010.

[Har77] Hartshorne R., *"Algebraic Geometry"*, New York: Springer-Verlag, chapter I.7, 1977.

[Has05] Hassouna M., Farag A., *"Robust Centerline Extraction Framework Using Level Sets"*, IEEE Conference on Computer Vision and Pattern Recognition, , 458-465, 2005.

[Hei94] Heisserman J., Woodbury R., *"Geometric design with boundary solid Grammars"*, Formal Design Methods for CAD, Amsterdam, 85–105, 1994.

[Hor90] Horn R., *"Matrix Analysis"*, Cambridge University Press, 1990.

[Hor94] Horn R., *"Topics in Matrix Analysis"*, Cambridge University Press, 1994.

[Jav10] *Java3D Home Site*, https://java3d.dev.java.net, 2010.

[Jog10] *Jogamp - Java libraries for 3D Graphics*, http://jogamp.org/jogl/www, 2010.

[Ken98] Kennedy S., Maestri G., Frantz R., *„3D Studio MAX. Czarna księga”*, Helion, 1998.

[Kni89] Knight T., *"Color grammars: designing with lines and colors"*, Environment and Planning B: Planning and Design 16(4) 417 - 449, 1989.

[Kni99] Knight T., *“Shape grammars in education and practice: history and prospects”*, The International Journal of Design Computing, part 2, 1999.

[Kor08] Korah T., Rassmussen C., *“Analysis for Building Textures for Reconstructing partially Occluded Facades”*, ECCV 2008: 10th European Conference on Computer, Springer, 361-370, 2008.

[Kou00] Koutamanis A., *“Representations from generative systems”*, Artificial Intelligence in Design - 2000, Kluwer, 225-244, 2000.

[Kri92] Krishnamurti R., Earl C., *"Shape recognition in three dimensions"*, Environment and Planning B: Planning and Design 19, 585–603, 1992.

[Kri92] Krishnamurti R., *"The maximal representation of a shape"*, Environment and Planning B: Planning and Design 19, 267-288, 1992.

[Kri92] Krishnamurti R., Earl C., *“Shape recognition in three dimensions”*, Environment and Planning, Planning and Design 19, 585-603, 1992.

[Kul03] Zenon K., *“From Picture Processing to Interval Diagrams”*, IFTR PAS Reports 4/2003, 2003.

[Lau04] Laurent S., *“Shaders for Game Programmers and Artists”*, Thomson Course Technology, 2004.

[Lia05] Li A., *“A Shape Set Grammar Interpretter with Thoughts of Emergence”* First international conference on design computing and cognition, Massachusetts Institute of Technology, 17–19, 2000.

[Llc10] Bools LLC (praca zbiorowa), *“Data Serialization Formats: XML, Soap, S-Expression, Abstract Syntax Notation One, Serialization, Yaml, Json, Lightweight Markup Language”*, General Books, 2010.

[Lue03] Luebke D., *"Level of detail for 3D graphics"*, Elsevier Science, 91-93, 2003.

[Mai02] Maierhofer S., *"Rule-Based Mesh Growing and Generalized Subdivision Meshes"*, Institute of Computer Graphics and Algorithms, Vienna University of Technology, PhD Thesis, 2002.

[Maj80] Majewski W., Albicki A., *„Algebraiczna teoria automatów"*, WNT, 14, 1980.

[Man96] Manfaat D., Duffy A., Lee B., *"Generalization of Spatial Layouts"*, Workshop on Machine Learning in Design, 23-24 June 1996, Fourth International Conference on Artificial Intelligence in Design'96, Stanford University, 1996.

[Mcu90] McCullough M., Mitchel W., *"The Electronic design studio: architectural knowledge and media in the Computer Era"*, Massachusetts Institute of Technology, 1990.

[Mit77] Mitchell W., *"Computer Aided Architectual Design"*, Charter Publisher N/York, 1977.

[Mit78] Mitchell W., Stiny G, *"The Palladian Grammar"*, Environment and Planning B 5,1078, 189-198, 1978.

[Mit96] Mitchell M., *"An introduction to genetic algorithms"*, Cambridge, MA, 1996.

[Mor05] Moraczewski I., *"Analyzing leaf margins with the use of a shape feature description language"*, Canadian Journal of Botany 76, 552-600, 2005.

[Mul96] Mulawka J., *"Systemy Ekspertowe"*, WNT Warszawa, 1996.

[Mus10] Muslimin R., *„Interweaving Grammar: Reconfiguring Vernacular Structure Through Parametric Shape Grammar"*, International Journal of Architectural Computing, 2010.

[Nie00] Nieson H., Nieson F., *"Shape analysis for mobile ambients"*, POPL '00 Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium On Principles Of Programming Languages, 2000.

[Ogl10] *OpenGL WebSite*, http://www.opengl.org, 2010.

[Ont04] *OntoSelect Ontology Library*, http://olp.dfki.de/ontoselect, 2004.

[Ont08] *Ontolingua*, http://www.ksl.stanford.edu/software/ontolingua, 2008.

[Ont10] *Protege Ontology Library*, http://protegewiki.stanford.edu/wiki/Protege_ Ontology_Library, 2010.

[Owl09] *W3C Recommendation for OWL language*, http://www.w3.org/TR/owl2-overview, 2009.

[Pan93] Praca zbiorowa, 1993, *„Nauka polska”*, Zakład Narodowy im. Ossolińskich, Oddzial w Warszawie, tom 41, 1-4, str 41-44

[Per07] Pérez A., López M., *“Ontological engineering: with examples from the areas of knowledge”*, Springer, 2007.

[Roz80] Rozenberg G., Salomaa A., *“The mathematical theory of L systems”*, Academic Press, New York, 1980.

[Nvi06] *Nvidia Corporation Articles*, http://www.nvidia.com/object/ IO_37226.html, 2006.

[Nvi07] *Nvidia Developer Articles*, http://http.developer.nvidia.com/CgTutorial/ cg_tutorial_chapter10.html, 2007.

[Shg07] *Shape Grammars*, http://shapegrammar.org, 2007.

[Shr04] Shreiner D., *“OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.1”*, Pearson Education, 40-47, 2004.

[Smi04] Smith B., *“Ontology and Information Systems”*, State University of New York at Buffalo, Department of Philosophy, Ontology, http://ontology.buffalo.edu/ontology(PIC).pdf, 2004.

[Sti06] Stiny G., *“Shape: Talking about Seeing and Doing*”, MIT Press, Cambridge, MA, 2006.

[Sti71] Stiny G. and Gips J., *"Shape grammars and the generative specification of painting and sculpture"*, Best of Computer Papers of 1971, 120-134, 1971.

[Sti72] Stiny G., Gips J., *“Shape grammars and the generative specification of painting and sculpture. Information Processing”*, Ed.C.V.Freiman (North-Holland, Amsterdam), 1460-1465, 1972.

[Sti80] Stiny G., *"Introduction to shape and shape grammars"*, Environment and Planning vol 1, 343-351, 1980.

[Sti82] Stiny G., *"Spatial Relations and Grammars"*, Environment and Planning, vol 9,113-115, 1982.

[Sti91] Stiny G., *"The algebras of design"*, Research in Engineering Design 2, vol 3, 171-181, 1991.

[Sti94] Stiny G., *"Shape rules: closure, continuity and emergence"*, Environment and Planning B: Planning and Design 21, 49–78, 1994.

[Stu07] Stouffs R., *"Sortal Structures: Supporting Representational Flexibility for Building Domain Processes"*, Computer-Aided Civil and Infrastructure Engineering 22, 98–116, 2007.

[Stu97] Stouffs R., Krishnamurti, R., *"Spatial change: Continuity, reversibility and emergent shapes"*, Environment and Planning B: Planning and Design, 24(3), 359-384, 1997.

[Sum04] Summers D., *„Texturing: Concepts and Techniques"*, Charles River Media Inc., 750-752, 2004.

[Tch00] Tchoń K., Mazur A., Dulęba I., Hossa R., Muszyński R., *"Manipulatory i roboty mobilne: Modele, planowanie ruchu, sterowanie"*, Warszawa, 2000.

[Tix10] *TinyXML HomePage*, http://www.grinninglizard.com/tinyxml, 2010.

[Tur05] Turek M., *"Zastosowanie standardu HL7 dla automatycznego pozyskiwania wiedzy i przetwarzania jej w drodze analizy analogii przyczynowo-skutkowych"*, Modelowanie Cybernetyczne Systemów Biologicznych, 74-86, 2005.

[Tur08] Turek M., *"Zmodyfikowane gramatyki kształtu w zastosowaniu dla generowania siatek 3d"*, Automatyka: półrocznik Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie t. 12 z. 3, 993–998, 2008.

[Tur09] Turek M., *„Optymalizacja skanowanych rotacyjnie siatek 3D prowadzona na potrzeby szybkiego renderowania w czasie rzeczywistym"*, Automatyka : półrocznik Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie t. 13 z. 3, 1469–1480, 2009.

[W3c08] *W3C Recomnedation for Extensible Markup Language (XML) 1.0 (Fifth Edition)*, http://www.w3.org/TR/REC-xml, 2008.

[WIN09] *Wings 3D Editor Home Site*, http://www.wings3d.com, 2009.

[Won03] Wonka P., Wimmer M., Sillion F., Ribarsky W., *"Instant architecture"*, ACM New York, NY, USA, 2003.

[Wri00] Wright R., Sweet M., *"OpenGL SuperBible"*, Waite Group Press, Tom 1, 299-304, 2000.

[Xsi10] *XSI Studio*, http://usa.autodesk.com/adsk/servlet/pc/index?id=13571168& siteID= 123112, 2010.

[Yvo03] Yvon G., Minich C., Perrin E., *„Boolean Operations on Feature-based Models"*, WSCG, 2003.

[Zho99] Zhou Y., Toga A., *"Efficient skeletonization of volumetric objects."*, IEEE Transactions on Visualization and Computer Graphics, Vol 5, 196-209, 1999.