# Projekt Looking Glass:

## Kompleksowy przegląd technologii

*Wyd. 0.2 14 Marca, 2006*

*Hideya Kawahara, Paul Byrne, Deron Johnson i Krishna Gadepalli*

*Projekt Looking Glass* http://lg3d-core.dev.java.net

*Sun Microsystems, Inc.* http://www.sun.com/

*Polskie tłumaczenie: Radosław Kierner, Stanisław Styszyński, Tomasz Kluza*

### Spis treści

1. Wprowadzenie..... 2
2. Zarys celów..... 2
3. Budowa i architektura..... 2
3.1 Główne cele projektowe..... 2
3.2 Realizacja..... 3
3.2.1 Wykorzystanie Javy i jej bogatego zbioru API..... 3
3.2.2 Oparcie na technologii Java 3D..... 3
3.2.3 Możliwość wykorzystania obiektów 3D różnych formatów..... 3
3.2.4 Model Klient-Serwer..... 3
3.2.5 Wykonywanie zachowań po stronie serwera..... 3
3.2.6 Obsługa zdarzeń..... 4
3.2.7 Interfejsy Event Adapter oraz Animation..... 4
3.2.8 Oparty na komponentach trójwymiarowy menadżer sceny..... 4
3.2.9 Abstrakcyjna warstwa integracji z natywnymi aplikacjami..... 4
3.2.10 Integracja z biblioteką Swing..... 4
3.3 Architektura wysokiego poziomu..... 5
4. API po stronie klienta..... 5
4.1 Klasy komponentów LG3D..... 5
4.2 Animacja transformacji..... 7
4.3 Klasy Event Adapter i Animation..... 7
4.3.1 Event Adapter..... 7
4.3.2 Action..... 7
4.3.3 Animacja..... 7
4.4 Przykładowe kody źródłowe..... 8
5. Przegląd środowiska LG3D..... 8
5.1 Menadżer Sceny - charakterystyka..... 8
5.1.1 Rozmieszczanie okien..... 8
5.1.2 Korzystanie z Tyłu Okna..... 9
5.1.3 Panoramiczny widok wielu pulpitów..... 9
5.1.4 Wykorzystanie przezroczystości do wyboru okna..... 9
5.2 Inkubator – Programy dla LG3D..... 9
5.2.1 Zoetrope – przeglądarka obrazów..... 11
5.2.2 Wybór tła pulpitu..... 11
5.2.3 Menadżer plików z wykorzystaniem głębi ekranu ( oś Z )..... 11
5.2.4 Solar System - Organizer/Kalendarz..... 11
6. Plany na najbliższą przyszłość..... 11
7. Miejsca warte odwiedzenia..... 11

# 1. Wprowadzenie

Projekt Looking Glass (LG3D) rozwijany jest na zasadach open source i oferuje użytkownikowi bogatsze wrażenia z pracy z komputerem i aplikacjami poprzez wizualizację trójwymiarową i jej możliwości. Wykorzystuje w pełni możliwości renderowania scen 3D współczesnych komputerów, które nie były dostępne gdy powstawało wiele dzisiejszych środowisk graficznych.

Celem niniejszego dokumentu jest opisanie ogólnej budowy środowiska Looking Glass, jego kluczowych elementów (używając prostych przykładów kodu, jeśli to konieczne), a także przegląd niektórych, najbardziej interesujących, efektów i aplikacji. Lektura tego dokumentu zapewni zrozumienie architektury wysokiego poziomu oraz API LG3D, a także wskaże perspektywy rozwoju, nad którymi potrzebna jest dalsza praca.

Ponieważ prace nad projektem Looking Glass wciąż trwają, pewne szczegóły wyspecyfikowane w tym dokumencie mogą ulec zmianie. Ostateczna forma API nie jest jeszcze ustalona, ale wciąż ewoluuje. Wszelkie sugestie na temat rozwoju środowiska są bardzo mile widziane poprzez korzystanie z oficjalnego forum dyskusujnego.

# 2. Zarys celów

LG3D przełamuje dwie granice w świecie doświadczeń użytkownika końcowego: dwuwymiarowość obecnych środowisk graficznych oraz sposób w jaki one ewoluują.

Ogólnie, praca wokół projektu skupia się na zapewnieniu solidnej platformy dla eksploracji trójwymiarowego środowiska typu desktop. W oparciu o to powstały manadżery okien oraz innowacyjne aplikacje w 3D. W celu umożliwienia poznania każdego aspektu technologii, udostępniliśmy kod LG3D społeczności , a nie tylko ograniczonej grupie programistów i testerów. Wierzymy, że otwarcie źródeł jest najlepszym sposobem współpracy.

Będąc platformą do eksploracji możliwości wizualizacji trójwymiarowej, Looking Glass nie jest tylko projektem laboratoryjnym. Aby uczynić LG3D w pełni konkurencyjną alternatywą dla istniejących rozwiązań, platforma wspiera integrację istniejących aplikacji w przestrzeni 3D oraz eksperymentalny menadżer okien 3D (Obraz 1). Na dzień dzisiejszy, integracja z natywnymi aplikacjami jest dostępna tylko dla systemów Linuks i Solaris x86. Platforma dla wytwarzania aplikacji 3D jest dostępna dla systemów Linuks, Solaris oraz Windows.

**Obraz 1: Programy 2D i 3D w LG3D**

# 3. Budowa i architektura

## 3.1 Główne cele projektowe

Platforma LG3D została zbudowana od zera w oparciu o doświadczenie zdobyte podczas implementacji wersji “Proof-of-Concept” (fazy 0 implementacji LG3D). Odzew na wersję proof-of-concept wskazał na następujące cztery aspekty kluczowe dla sukcesu wersji trial:

(1) integracja z istniejącymi aplikacjami 2D,
(2) Dwu-i-pół-wymiarowa obsługa, analogiczna do obecnych środowisk wykorzystująca jednak w pełni zalety 3D,
(3) Rozbudowana interakcja z użytkownikiem,
(4) Reprezentacja wizualna.

Nauczyliśmy się także, że interfejs użytkownika środowiska 3D to kompletnie nowy obszar i wymaga dalszych eksperymentów. Biorąc wszystkie te aspekty pod uwagę, zdefiniowaliśmy następujące cele projektowe:

- Silne wsparcie dla odkrywania możliwości trójwymiarowego interfejsu użytkownika
- Stabilna platforma dla produktów na niej bazujących
- Wysoka wydajność i skalowalność
- Silne wsparcie dla technologii 3D
- Możliwość uruchamiania istniejących aplikacji 2D
- Wsparcie dla dwu-i-pół-wymiarowego menadżera okien
- Bogaty zbiór bibliotek
- Bogata interakcja z użytkownikiem oparta na mechanizmie animacji
- Wsparcie dla programowania wizualnego

## 3.2 Realizacja

W celu osiągnięcia celów projektowych opisanych w poprzednim rozdziale, następujące zagadnienia zostały, lub będą, zaimplementowane:

- Wykorzystanie Javy i jej bogatego zbioru API
- Oparcie na technologii Java 3D
- Możliwość wykorzystania obiektów 3D różnych formatów
- Model Klient-Serwer
- Wykonywanie zachowań po stronie serwera
- Obsługa zdarzeń
- Interfejsy Event Adapter oraz Animation
- Oparty na komponentach menadżer okien
- Abstrakcyjna warstwa integracji z istniejącymi aplikacjami
- Integracja z biblioteką Swing

### 3.2.1 Wykorzystanie Javy i jej bogatego zbioru API

Język Java został wybrany jako podstawowy język programowania, ponieważ gwarantuje wysoką produktywność i bezpieczeństwo, dostarcza także bogaty zbiór bibliotek do wykorzystania. Korzyści te pozwalają na pełną eksplorację nowego środowiska 3D.Ponadto, dzięki komercyjnym zastosowaniom w sferze serwerów, jej efektywność została znacząco poprawiona. Wersja "proof-of-concept" dowiodła, że efektywność technologii Java nie jest problemem. Mimo, iż obecnie tylko zbiór API języka Java jest wspierany, w przyszłości, interakcja z C++ także zostanie wprowadzona w celu uruchamiania natywnych aplikacji w tym języku w LG3D.

### 3.2.2 Oparcie na technologii Java 3D

API i implementacja LG3D opiera się na technologii Java 3D.Ponadto, platforma dostarcza dodatkową funkcjonalność jak architekturę komponentów oraz system animacji, w celu uproszczenia tworzenia wysoce interaktywnych aplikacji.Java 3D, wprowadzona w 1997 roku, jest dojrzałą, opartą na języku Java technologią dla wysokiej jakości, skalowalnego, niezależnego od platformy, renderingu 3D. API Java 3D dostarcza zbiór zorientowanych obiektowo interfejsów wspierających prosty model programowania wysokiego poziomu, który może służyć budowie, renderowaniu i kontroli zachowań obiektów i światów 3D. Zarazem, wykorzystuje w pełni akceleracją sprzętową renderingu 3D. Opierając się na niej, LG3D może korzystać ze wszystkich powyższych jej zalet. Dodatkowo, możemy wykorzystać wszystkie zgromadzone zasoby jak podrzęczniki, książki i biblioteki, jak mechanizm wczytywania modeli 3D. To czyni platformę produktywnym środowiskiem programistycznym a także solidną i wysoce skalowalną.

Mimo iż platforma opiera się na Java 3D, została podjęta decyzja o wykorzystaniu części API Java 3D po stronie klienta.Elementy komplikujące implementację zostały usunięte. Wszystkie metody o typie double zostały usunięte lub zastąpione wersjami opartymi na typie float. Pozwoli to w przyszłości na uruchamianie aplikacji klienckiej w ograniczonym sprzętowo środowisku jak terminal do telewizora (set top box) lub samochodowy system nawigacji.

### 3.2.3 Możliwość wykorzystania obiektów 3D różnych formatów

Wsparcie dla różnych formatów obiektów 3D uważamy za kluczową funkcję technologii LG3D, gdyż zaprasza ona grafików do procesu wytwarzania oprogramowania. W tym celu został wprowadzony specjalny komponent LG3D o nazwie `ModelLoader`. Oferuje on prosty, ale rozbudowany mechanizm wczytywania modelu 3D z pliku i umiejscowienie go w środowisku LG3D. Obsługuje wiele formatów poprzez użycie mechanizmu obsługi modeli trójwymiarowych w Java 3D.

### 3.2.4 Model Klient-Serwer

Model Klient-Serwer pozwala wielu procesom na współdzielenie tego samego wirtualnego świata 3D. Ponadto technologia ta pozwala platformie na uruchamianie aplikacji 3D dla LG3D jako oddzielny proces (nawet na innej maszynie), podczas gdy interfejs użytkownika jest wyświetlany w świecie 3D, w którym użytkownik pracuje. Tak jak model klient-serwer w środowisku X, pozwala to na izolację aplikacji w ramach platformy.Ten aspekt jest wciąż w stanie tworzenia i nie jest w pełni zaiplementowany. Należy zwrócić uwagę, że termin K*lient-Serwer* niekoniecznie oznacza, że klient i serwer znajdują się na różnych maszynach połączonych siecią, ale raczej działają na tej samej maszynie (ale nie muszą). Klient to aplikacja wysyłająca żądanie renderowania jej elementów interfejsu użytkownika. Zazwyczaj, na maszynie pracuje pojedynczy serwer odbierający żądanie renderingu od klientów i konstruuje obraz pulpitu.

### 3.2.5 Wykonywanie zachowań po stronie serwera

Animacja zazwyczaj operuje na obiektach graficznych w każdej klatce jej trwania. Mechanizm wykonujący animację po stronie serwera został wprowadzony, aby wyeliminować konieczność komunikacji między klientem a serwerem w każdej klatce animacji. Klient

określa typy animacji poprzez parametry, takie jak czas trwania. Platforma dostarcza wiele typów predefiniowanych animacji. Dodatkowo, animacja może być zrealizowana poprzez wykorzystanie obsługi animacji w mechanizmie wczytywania modeli. Z takimi możliwościami, ograniczenia wynikające z obsługi zachowań po stronie serwera (np. trudność z obsługą zachowań klienta) nie są uważane za krytyczne.Mechanizm ten nie jest obecnie w pełni zaimplementowany.

### 3.2.6 Obsługa zdarzeń

Bogaty wachlarz interakcji z użytkownikiem i modularyzacja biblioteki były dwoma kluczowymi wymaganiami dla platformy. W celu implementacji interakcji z użytkownikiem, znacząca ilość informacji o stanach i zdarzeniach musi być rozesłana po systemie. To mogłoby spowodować nadmierną zależność między komponentami jeśli opieralibyśmy się całkowicie na wywoływaniu metod. Asynchroniczny system obsługi zdarzeń jest wspierany na poziomie platformy w celu dekompozycji biblioteki na mniejsze części, które mogą ze sobą współpracować bardziej swobodnie i niezależnie.

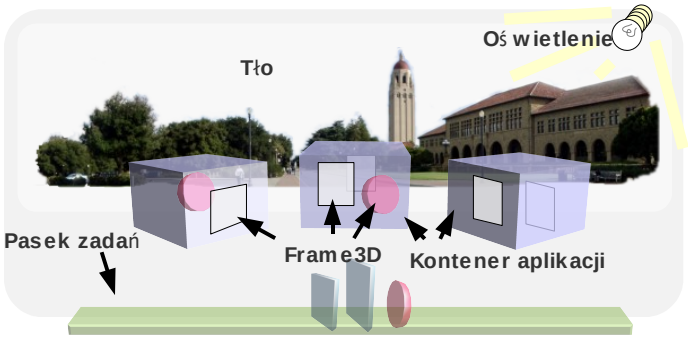
### 3.2.7 Interfejsy Event Adapter oraz Animation

W oparciu o sugestie użytkowników wersji “proof-of-concept”, interakcja z użytkownikiem została uznana za jeden z kluczowych elementów. Najczęściej realizowanym wywołaniem w implementacji interakcji z użytkownikiem jest odbieranie zdarzeń myszy i wykonywanie animacji, jak zmiana rozmiaru przycisku pod kursorem myszy. Platforma wprowadza abstrakcyjne interfejsy `EventAdapter` oraz `Action`które separują obsługę zdarzeń od implementacji animacji. Pozwala to łączyć komponenty w celu prostszej realizacji interakcji z użytkownikiem.Szczegóły API są opisane w Rozdziale 4.3.

W przyszłości, zostanie wprowadzony system animacji oparty na zdarzeniach z prostą kompozycją zachowań. Pozwoli on na łączenie komponentów bez pisania dodatkowego kodu.

### 3.2.8 Oparty na komponentach trójwymiarowy menadżer sceny

Menadżer Sceny (Scene Manager) pełni rolę menadżera okien dla przestrzeni 3D, ponieważ interfejs typowej aplikacji dla LG3D posiada unikalne kształty nie ograniczone przez kształt okna. Menadżer Sceny wchodzi w interakcję z użytkownikiem i aplikacjami. Implementuje dostosowywalną politykę zarządzania aplikacjami działającymi w środowisku i jest odpowiedzialny za ich aranżację w przestrzeni 3D.




**Rys 1: Główne komponenty menadżera sceny**

Aby stymulować pomysły, został stworzony wstępny framework dla opartego na komponentach Menadżera Sceny. W jego implemetacji, wzięto pod uwagę zarówno dwu-i-pół jak i trójwymiarową obsługę. Rys 1 powyżej pokazuje główne komponenty Menadżera Sceny: kontener aplikacji, tło, oświetlenie globalne oraz pasek zadań.

### 3.2.9 Abstrakcyjna warstwa integracji z natywnymi aplikacjami

Aby uczynić LG3D dostępnym dla różnych platform, została zaimplementowana abstrakcyjna warstwa dla integracji z natywnymi aplikacjami.

Warstwę tą możemy podzielić na dwie główne części : moduł Foundation Window System (FWS) oraz moduł Native Window Representation.Moduł FWS zapewnia przechwytywanie obrazu aplikacji i dostarczanie zdarzeń z i do natywnego systemu okien. Moduł ten posiada zbiór interfejsów dla implementacji wtyczek dla różnych natywnych systemów okien.

Obecnie istnieją dwie implementacje, wersja dla X11 oraz wersja dla AWT. Integracja z X11 pozwala na uruchamianie i wyświetlanie aplikacji X na platformie LG3D. Wersja AWT pozwala na uruchamianie platformy LG3D w systemach operacyjnych wspierających Javę oraz Javę 3D ( jak Solaris czy Windows), lecz bez integracji z natywnymi aplikacjami. Moduł Native Window Representation jest umiejscowiony na szczycie FWS i zapewnia abstrakcyjną reprezentację elementów wizualnych natywnej aplikacji w przestrzeni 3D.

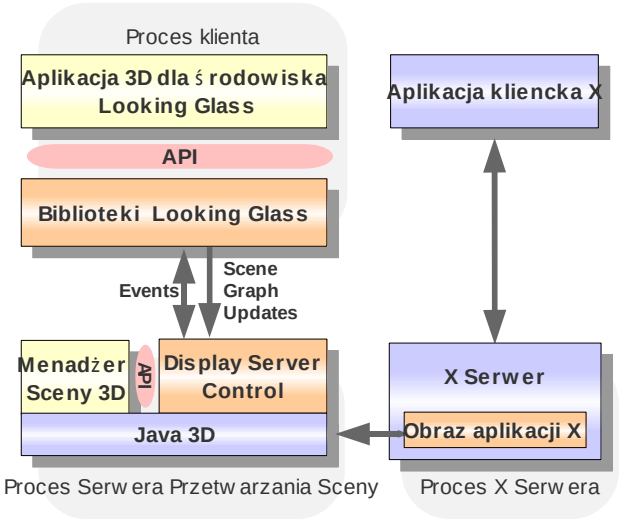
### 3.2.10 Integracja z biblioteką Swing

Bazując na doświadczeniu zdobytym podczas tworzenia aplikacji przy użyciu prototypowego API, oraz na podstawie sugestii ze strony społeczności, stało się jasne, że mechanizm umożliwiający programistom na użycie biblioteki Swing jest rzeczą bardzo ważną.Nawet tworząc aplikację w 3D, czasami

standardowe, dwuwymiarowe komponenty graficzne są bardzo przydatne. Przykładem może być okno dialogowe wyboru pliku lub pole wprowadzania tekstu. Zamiast wprowadzać całkowicie nowy zbiór API, LG3D oferuje mechanizm integracji komponetów Swing ze środowiskiem.Specjalny komponent o nazwie `SwingNode` został stworzony właśnie do tego celu. Jest to węzęł do którego można dołączyć wchodzący w skład biblioteki Swing `JPanel`. Programiści mogą skonstruować `JPanel`używając standardowego Swing API, dołączyć go do `SwingNode`, a następnie dołączyć `SwingNode` do aplikacji 3D. W ten sposób, dwuwymiarowy komponent `JPanel` może występować w przestrzeni trójwymiarowej, a na aplikacji takiej można wykonywać operacje używając punktów odniesienia w przestrzeni 3D, tak samo jak w przypadku standardowych kompenentów LG3D.

## 3.3 Architektura wysokiego poziomu

Architektura wysokiego poziomu LG3D jest pokazana na Rysunku 2. Składa się z dwóch głównych części: realizującej integrację z X11 (prawa strona wykresu) oraz odpowiedzialnej za obsługę trójwymiarowych aplikacji dla LG3D. Aplikacja kliencka, niemodyfikowana w żaden sposób, komunikuje się z X Serwerem, który jest rozszerzony o mechanizm przechwytywania aplikacji oraz inne niezbędne funkcje. Rozszerzony X Serwer przechwytuje wizualną reprezentację klienta i przesyła ją do serwera przetwarzającego scenę (LG3D Display Server), który zarządza renderowaniem przestrzeni 3D, używając platformy Java 3D. Wszystkie rozszerzenia są integrowane z X serwerem X.org i będą domyślnie dostępne.




**Rys 2: Architektura wysokiego poziomu**

Dla ułatwienia tworzenia aplikacji dla środowiska LG3D, platforma dostarcza zbioru bibliotek LG3D. Jest to pokazane na lewej górnej części schematu. Biblioteki komunikują się z Serwerem Przetwarzania Sceny w celu konstrukcji obiektów wizualnych 3D oraz obsługi zdarzeń. API dla tworzenia aplikacji w LG3D jest nazywane API po stronie klienta (Client-side API). Zostało ono szczegółowo opisane w następnym rozdziale.

# 4. API po stronie klienta

Jednym z celów projektowych API po stronie klienta było to, aby nie było trzeba dysponować rozległą wiedzą na temat programowania w 3D, by z niego korzystać.

API po stronie klienta może zostać podzielone na następujące dwa główne obszary:

- klasy definiujące obiekty graficzne, oparte na Java 3D
- specjalizowane klasy obsługujące 3D

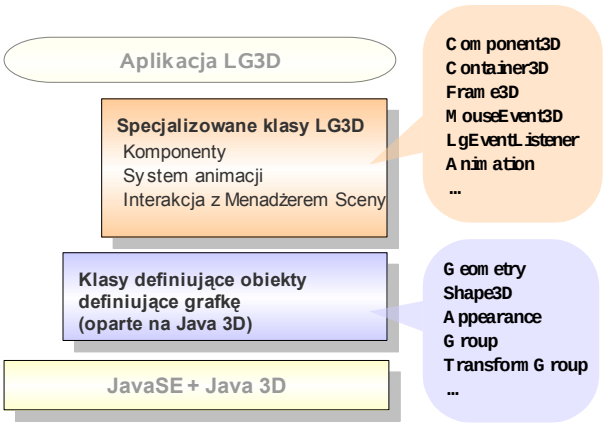
Rys. 3 (na następnej stronie) przedstawia ogólny przegląd API, natomiast Rys. 4 pokazuje bardziej szczegółowo związki pomiędzy większością głównych klas.

## 4.1 Klasy komponentów LG3D

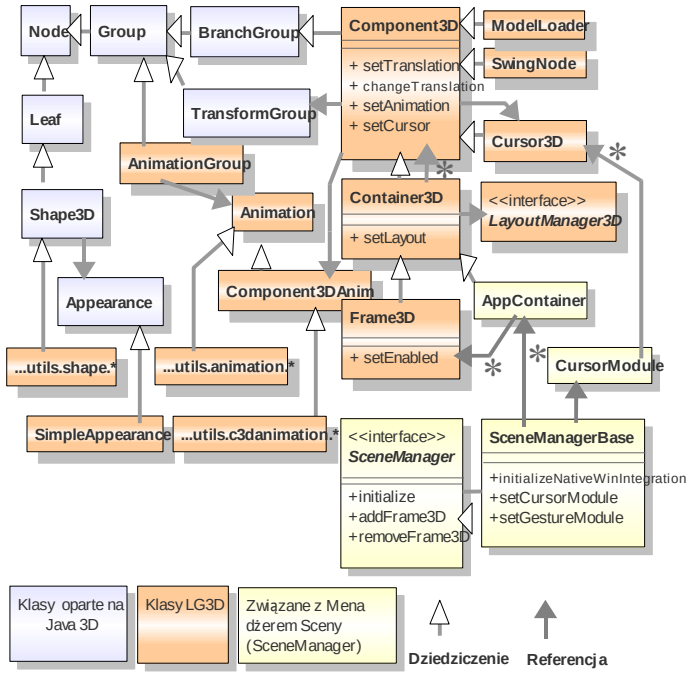
Klasy wchodzące w skład pakietu `org.jdesktop.lg3d.wg` definiują podstawowe funkcje wykorzystywane do tworzenia komponentów w środowisku LG3D. Prace projektowe są wciąż w początkowym stadium i będą kontynuowane w najbliższej przyszłości w celu dalszego udoskonalania.

Tabela 1 przedstawia główne klasy wchodzące w skład opisywanego pakietu. Klasy te są analogiczne do klas znanych z biblioteki AWT, jedyną różnicą jeśli chodzi o nazewnictwo jest końcówka "3D."

Fragment kodu na listingu 1 przedstawia bardzo prostą aplikację wykorzystującą główne klasy LG3D. Efektem działania aplikacji jest wyświetlenie kolorowego sześcianu.




**Rys. 3: Przegląd API LG3D po stronie klienta**




**Rys. 4: Związki między klasami API po stronie klienta**

```
public class LG3SimpleDApp {
  public LG3DApp() {
    Frame3D frame = new Frame3D();
    ColorCube cube = new ColorCube(0.05f);
    Compnent3D comp = new Component3D();
    comp.addChild(cube);
    comp.setCursor(Cursor3D.SMALL_CURSOR);
    comp.addChild(comp);
    frame.changeVisibile(true);
    frame.changeEnabled(true);
  }
}
```

**Listing 1: Użycie głównych klas**

**Tabela 1: Główne klasy LG3D UI**

| *Klasa* | *Opis* |
|---|---|
| `Component3D` | Nadrzędna superklasa wszystkich komponentów. Wspiera proces wizualizacji poprzez dodawanie węzłów potomnych definiujących grafikę, stworzonych przez użycie, opartego na Java 3D API. Należy podkreślić, że zaimplementowano ograniczenie niepozwalające na dołączanie obiektu `Component3D` do innego obiektu typu `Component3D` jako komponentu pochodnego. W takich przypadkach należy użyć obiektu `Container3D` jako komponentu nadrzędnego. Klasa ta stanowi źródło zdarzeń i implementuje interfejs `LgEventSource`. Obsługa zdarzeń myszy zostanie opisana bardziej szczegółowo na następnych stronach. Klasa ta zawiera także wbudowane wsparcie dla operacji transformacji. |
| `Container3D` | Rozszerzenie klasy `Component3D`. Obiekt typu `Container3D` jest komponentem umożliwiającym dołączanie komponentów pochodnych, oprócz obiektów definiujących grafikę opartych na Java 3D. Zazwyczaj, taki obiekt jest opakowany przez obiekt `Component3D` i dopiero wtedy może być dodany do obiektu `Container3D`. Klasa ta jest zazwyczaj używana razem z interfejsem `LayoutManager3D`. |
| `Frame3D` | Rozszerzenie klasy `Container3D`. Odpowiednik klasy Frame w bibliotece AWT: kontener główny dla wszystkich aplikacji dla środowiska LG3D. Na przykład, wymienia się informacjami z Menadżerem Sceny w celu rozmieszczenia obiektów w przestrzeni. |
| `Cursor3D` | Rozszerzenie klasy `Component3D` stanowiące klasę nadrzędną dla wszystkich kursorów. Aplikacja może definiować własne kursory poprzez rozszerzanie tej klasy, albo poprzez jeden z konstruktorów przyjmujący obiekt `Component3D` jako implementację kursora. Dodatkowo pozwala na dodawanie i pobieranie kursorów, a także na pracę z predefiniowanymi kursorami obsługiwanymi przez Menadżera Sceny. |

**Tabela 2: Metody animacji transformacji**

| *Rodzina metod `set*()`* | *Rodzina metod `change*()`* |
|---|---|
| `setTranslation()` | `changeTranslation()` |
| `setRotationAngle()` | `changeRotationAngle()` |
| `setScale()` | `changeScale()` |
| `setVisible()` | `changeVisibile()` |

## 4.2 Animacja transformacji

Klasa `Component3D` posiada dwa zestawy metod służacych transformacji. Jednym z nich jest rodzina metod `set*()` a drugim rodzina metod `change*()` (zob: Tabela 2).Metody z rodziny `set*()` dokonują przekształceń natychmiastowo, natomiast metody z rodziny `change*()` mogą je wykonywać z pewnym opóźnieniem. Na przykład, użycie metody `setScale(2.0f)` zmienia skalę na 2.0 od razu, podczas gdy użycie `changeScale(2.0f, 1000)` potrzebuje 1000 milisekund na wykonanie płynnej animacji przejścia. System animacji jest całkowicie zmodularyzowany. Wyzwalacz wywołuje zdarzenie odpowiadające danej sytuacji.

Pozwala to na uproszczenie implementacji płynnego ruchu komponentów, który można zobaczyć w prezentacji demonstrującej możliwości LG3D.

## 4.3 Klasy Event Adapter i Animation

Jak to było wspomniane w rozdziale 3.2.7, platforma definiuje abstrakcyjne interfejsy `EventAdapter` oraz `Action`, rozdzielające obsługę zdarzeń od implementacji animacji. Pozwala to na łączenie komponentów w celu prostszej interakcji z użytkownikiem .Komponenty te wchodzą w skład pakietu `org.jdesktop.lg3d.utils` i stanowią główną część zbioru klas użytkowych LG3D.

Rys 5 przedstawia związki między klasami `EventAdapter`, `Action` oraz `Animation`.

### 4.3.1 Event Adapter

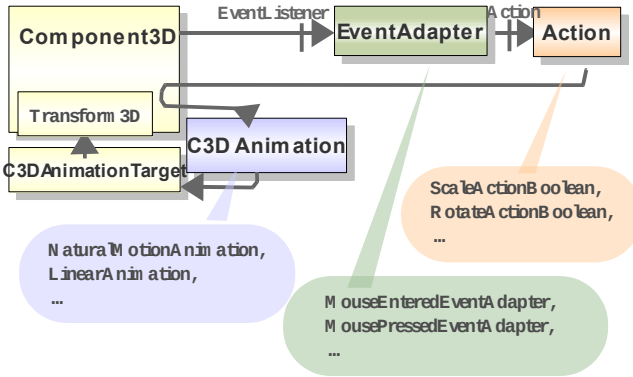
Obiekt `EventAdapter` jest odpowiedzialny za odbieranie zdarzeń. Upraszcza informacje uzyskane przez zdarzenie i wywołuje metodę `performAction` skojarzoną z obiektem `Action` lub `ActionAdapter`. Klasy wchodzące w skład podpakietu `eventadapter` implementują interfejs `LgEventListener` i muszą posiadać przynajmniej jedną referencję na obiekt implementujący interfejs `Action`. Biblioteka LG3D zawiera predefiniowane klasy adaptacyjne obsługujące niektore zdarzenia ( z ang. `EventAdapter's` ) takie jak `MouseEnteredEventAdapter` oraz `MousePressedEventAdapter`.

### 4.3.2 Action

Klasy wchodzące w skład podpakietu `action` implementują interfejs `Action`. Otrzymują informacje przetworzone przez obiekt `EventAdapter` poprzez wywołanie metody `performAction` i realizują wizualną odpowiedź na żądanie. Obiekt `Action` zazwyczaj posiada referencję na obiekt `Component3D` aby móc wykonać wizualizację danej animacji poprzez użycie metod z rodziny `change*()` opisanej w rozdziale 4.2. Predefiniowanymi akcjami (`Action`s) są `ScaleActionBoolean` oraz `RotationActionBoolean`.

### 4.3.3 Animacja

Klasy definiujące animacje rozszerzają abstrakcyjną klasę `Animation`w celu implementacji polityki animacji, na przykład akceleracji i deakcelaracji (zadawania opóźnienia) ruchów. Istnieją dwa rodzaje wezłów w LG3D do których można podpiąć animację. Jednym jest `AnimationGroup`, który jest węzłem sceny z którym animacja może być skojarzona, drugim jest `Component3D`. Używamy klasy `Component3DAnimation` , która rozszerza klasę `Animation` o implementację animacji komponentów. Rodzina metod `change*()` opisana w rozdziale 4.2 używa określonych metod animacji komponentów w celu wykonania transformacji. Predefiniowanymi animacjami komponentów są `NaturalMotionAnimation` oraz `LinearAnimation`.




**Rys 5: Klasy EventAdapter, Action oraz Animation**

## 4.4 Przykładowe kody źródłowe

Następujący fragment kodu (Listing 2) przedstawia szablon aplikacji dla LG3D. Po pierwsze, musimy stworzyć nadrzędny kontener dla aplikacji 3D. Do tego celu służy klasa `Frame3D`. Programiści zazwyczaj korzystają z klas definiujących obiekty graficzne w celu tworzenia kształtów obiektów 3D, ich wyglądu i przypisania ich do komponentu. Jeśli to konieczne, dołącza się do komponentu animację i listenery. Po wstępnej inicjalizacji, interfejs graficzny jest udostępniany poprzez wywołanie metod `frame.changeVisibe(true)` oraz `frame.changeEnabled(true)`.

```
    Frame3D frame = new Frame3D();
    Container3D container = new Container3D();
    Component3D comp = new Component3D();
    ...
    // Tworzenie nowego kształtu – definiowanie
//geometrii obiektu, wyglądu, tekstur etc.
    comp.addChild(aShape);
    ...
    // Inicjalizacja animacji
    comp.setAnimation(anAnimation);
    ...
    // Inicjalizacja obsługi zdarzeń
    comp.addListener(anEventListener);
    ...
    container.addChild(comp);
    frame.addChild(container);
    frame.changeVisibe(true);
    frame.changeEnabled(true);
```

**Listing 2: Szablon aplikacji dla środowiska LG3D.**

```
Component3D comp = new Component3D();
// Inicjalizacja i dodanie kształtu
SimpleAppearance app
  = new SimpleAppearance(0.6f, 0.8f, 0.6f);
Box box = new Box(0.04f, 0.03f, 0.02f, app);
comp.addChild(box);

// Inicjalizacja animacji
comp.setAnimation(new NaturalMotionAnimation(1000));

// Inicjalizacja obsługi zdarzeń
comp.addListener(
  // Gdy przycisk myszy jest wciśniety...
  new MousePressedEventAdapter(
    // obróć komponent o 180 stopni
    new RotateActionBoolean(comp,(float)Math.PI)));
```

**Listing 3: Użycie klas użytkowych**

Powyższy fragment kodu przedstawia przykład użycia klas użytkowych w LG3D. Wynikiem jest stworzenie sześcianu przy użyciu klas LG3D definiujących grafikę i dodanie go do komponentu. Następnie ustawiana jest animacja komponentu imitująca naturalny ruch. Domyślny czas trwania animacji tranformacji jest ustawiony na `1000` milisekund poprzez podanie argumentu do konstruktora.

Następnie, ustawia się `MousePressedEventAdapter` jako odbiorcę zdarzenia. Metoda `processEvent()` jest wywoływana zawsze, gdy przycisk myszy jest wciskany lub zwalniany. Wywoływana jest wtedy metoda `performAction()` ,którą definiuje obiekt typu action, podany jako argument. W omawianym przykładzie jest to obiekt `RotateActionBoolean`.

`RotateActionBoolean`implementuje metodę `performAction()`w celu rotacji danego komponentu (`comp`) poprzez podanie kąta (`((float)Math.PI`), gdy przycisk myszy pozostaje kliknięty. Aby to osiągnąć wywoływana jest metoda komponentu `changeRotationAngle()`. Domyślnie, wektor osi obrotu wynosi `(0, 1, 0)`.

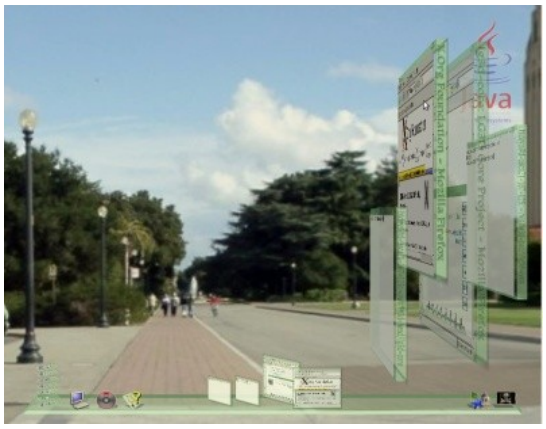
# 5. Przegląd środowiska LG3D

## 5.1 Menadżer Sceny - charakterystyka

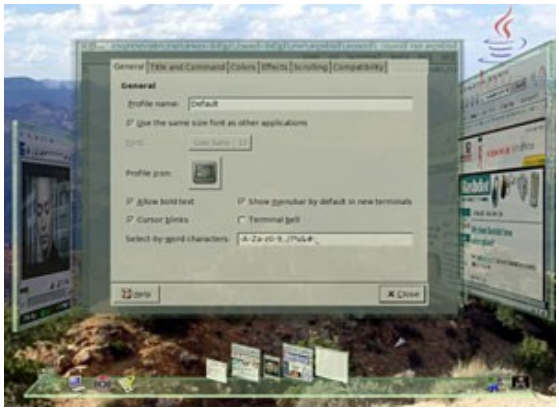
Menadżer Sceny został zaimplementowany z wykorzystaniem opisanego wyżej API. Ponieważ w środowisku LG3D okno faktycznie jest obiektem 3D, pracę z Menadżerem Sceny cechuje duża swoboda rozmieszczania i możliwych układów okien oraz stosowania efektów wizualnych. Wyzwanie stanowi zaproponowanie rozwiązania użytecznego, oryginalnego i jednocześnie atrakcyjnego wizualnie, które wzbogaci dotychczasowy sposób pracy z tradycyjnym pulpitem, nie wymagając jednak zmiany dotychczasowych przyzwyczajeń. Poniżej zaprezentowano niektóre, uznane za najbardziej rewolucyjne i atrakcyjne, możliwości Menadżera Sceny.

### 5.1.1 Rozmieszczanie okien

Jednym z naszych zamierzeń jest pozwolić użytkownikowi obracać i odkładać okna na bok. Jest to przydatna opcja, gdy użytkownik nie pracuje bezpośrednio z oknem, ale chce mieć je na oku, np. aby nadzorować zmiany w tym oknie. (Obraz 1). Funkcja ta pozwala lepiej dysponować ograniczonym obszarem pulpitu podczas pracy z wieloma oknami. Poza tym możemy ustawić w ten sposób wszystkie okna jednocześnie, aby móc zobaczyć komplet okien znajdujących się na pulpicie. Nazwaliśmy tą funkcję Biblioteczką. (Obraz 2). Każde okno posiada tytuł na grzbiecie (podobnie jak książki), dla wygody użytkownika.



**Obraz 2: Biblioteczka Okien**



**Obraz 3: Konfiguracja z wykorzystaniem tyłu Okna**


**Obraz 4: Panoramiczny widok wielu pulpitów**

### 5.1.2 Korzystanie z Tyłu Okna

Ponieważ okno jest obiektem 3D, możemy zagospodarować jego tylnią stronę. Stanowi ona nową wartość w przezwyciężaniu ograniczenia rozmiaru ekranu. Wprowadziliśmy prototypowy mechanizm korzystania z tzw. “żółtych karteczek” z notatkami użytkownika, przyklejanych z tyłu okna z programem. Poza tym projektujemy i wprowadzamy konfigurację programu z wykorzystaniem drugiej strony okna (Obraz 3).

### 5.1.3 Panoramiczny widok wielu pulpitów

Gdy użytkownik wykorzysta przestrzeń dostępną w ramach pulpitu, zapewne chciałby przesunąć pulpit na bok i uzyskać kolejny pulpit do zagospodarowania. Wprowadziliśmy taką możliwość w LG3D. Rozwiązanie to przypomina przełączanie pulpitów, ale poprzez powiązanie zmiany pulpitu z panoramicznym tłem i płynnym przejściem miedzy poszczególnymi pulpitami, uzyskano lepsze wrażenie wizualne (Obraz 4). Tło pulpitu może być dowolną zrenderowaną sceną pełnego modelu 3D. Dla przykładu, można wykorzystać zrenderowaną scenę typowego domu i podział na salon lub pokój gościnny ( wszystko związane z rozrywką, umieszczone zostaję w obrębie tego pulpitu ), pokój do nauki i pracy, garaż ( wszystko związane z narzędziami systemowymi ) itd. Dzięki środowisku 3D wprowadzenie takiego podziału przebiega intuicyjnie, a korzystanie z niego polega na zmianie położenia punktu, z którego patrzymy na pulpit ( odpowiadające przejściu do innego pokoju ). Panoramiczny pulpit wyposażony jest również w tryb podglądu poszczególnych pulpitów.

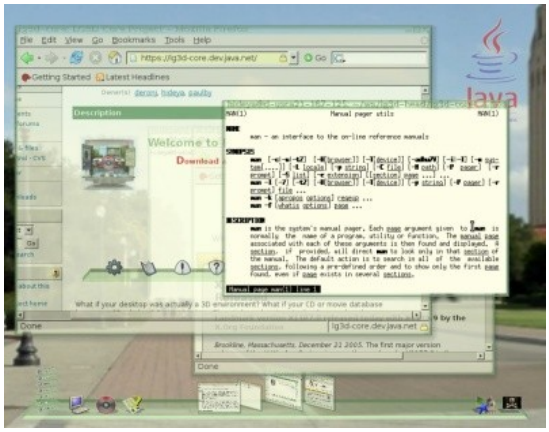
### 5.1.4 Wykorzystanie przezroczystości do wyboru okna

Wyobraź sobie pulpit pełen otwartych okien. Czy nie byłoby miło móc łatwo spojrzeć na dowolne, nawet przesłonięte przez inne programy, okno ? LG3D pozwala użytkownikowi to osiągnąć poprzez przesunięcie kursora nad miniaturkę okna. W tym samym czasie pozostałe okna stają sie prawie przezroczyste ( Obraz 5, 6 ). Dzięki temu użytkownik widzi głównie wybrane okno. W ten sposób, dzięki zmianie położenia kursora w obszarze miniaturek otwartych okien, użytkownik może szybko znaleźć potrzebne okno.

## 5.2 Inkubator – Programy dla LG3D

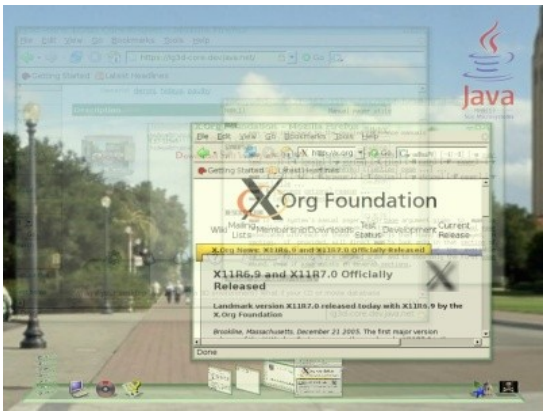
Inkubator jest miejscem dla członków społeczności LG3D wspierającym tworzenie i udostępnianie programów korzystających z LG3D, zanim zostaną w pełni zintegrowane z podstawowym projektem. Obecnie prowadzonych jest około dwudziestu zarejestrowanych projektów, niektóre z nich już zostały wcielone do wydanych wersji LG3D.

Poniższa sekcja prezentuje niektóre z interesujących programów rozwijanych przez społeczność LG3D w ramach Inkubatora. Pragniemy zauważyć, że są to ciągle wersje eksperymentalne programów.

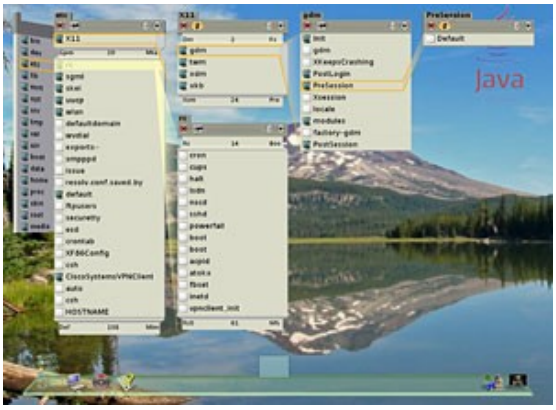


**Obraz 5: Wykorzystanie przezroczystości do wyboru Okna (1)**

**Obraz 8: Zarządzanie wyborem tła**



**Obraz 6: Wykorzystanie przezroczystości do wyboru Okna (2)**



**Obraz 9: Menedżer plików z wykorzystaniem głębi ekranu ( oś Z )**

**Obraz 7: Zoetrope - przeglądarka obrazów**

**Obraz 10: Solar System - Organizer/Kalendarz**

### 5.2.1 Zoetrope – przeglądarka obrazów

To przeglądarka obrazów z drobnym, ale efektownym wykorzystaniem przestrzeni (Obraz 7). Po lewej stronie umieszczono miniaturki obrazów w układzie koła Ferrisa (inaczej koła diabelskiego, spotykanego np. w Wesołych Miasteczkach). Ruch rolki myszki powiązany jest z kołem minaturek. Pozwala to na bardzo intuicyjny i łatwy wybór obrazków.

### 5.2.2 Wybór tła pulpitu

Program Background Manager pozwala użytkownikowi zmieniać tło pulpitu (Obraz 8). Dostępnych jest 50 obrazów tła. Miniaturki rozmieszczone zostają przez program w przestrzeni 3D w taki sposób, aby dostarczyć użytkownikowi wstępnej informacji o każdym tle i jednocześnie nie zabrać zbyt dużo miejsca na pulpicie.

### 5.2.3 Menadżer plików z wykorzystaniem głębi ekranu ( oś Z )

Ten program również korzysta z trójwymiarowości środowiska LG3D w ograniczony, ale mądry sposób (Obraz 9). Wraz z otwieraniem podkatalogów przez użytkownika, drzewko rozbudowuje się na prawo. Kiedy osiągnie prawy skraj ekranu, początek drzewa obraca się bokiem do użytkownika zamiast ukrywać się. Taki widok ciągle pozwala na zapoznanie się z zawartościa danego poziomu. Gdy użytownik umieści kursor nad takim fragmentem drzewa, okno obraca się ponownie w jego kierunku. Wtedy inny, nieużywany fragment widoku obraca się bokiem po to, aby prawa krawędź widoku nie przekroczyła krawędzi ekranu.

### 5.2.4 Solar System - Organizer/Kalendarz

Ostatni przykład stanowi prawdziwe wyzwanie. Kiedy myślimy o programie-organizerze, przeważnie mamy na myśli konstrukcję wykorzystującą tradycyjna postać kalendarza. Autorzy tego projektu zastanawiali sie w jaki sposób odejść od tego schematu. Postanowili wykorzystać model Układu Słonecznego jako motyw (Obraz 10).

Każda orbita reprezentuje typ zadania, a planeta konkretne zadanie. Wzdłuż orbit przepływa oś czasu – z przeszłości w przyszłość. Im bardziej ważne jest zadanie, tym większa planeta je reprezentuje. Jest to kreatywny i nie schematyczny przykład wykorzystania możliwości drzemiących w środowisku 3D.

Projekty przedstawione powyżej znajdują się nadal w fazie produkcyjnej, ale nawet teraz wspaniale jest móc widzieć, jak Projekt Looking Glass stanowi platformę, na której następuje ich rozwój w kierunku programów 3D.

## 6. Plany na najbliższą przyszłość

Obecnie, koncentrujemy się na poprawie stabilności i funkcjonalności oraz łatwej do przeprowadzenia instalacji. Pracujemy ze społecznością Ubuntu, aby uczynić LG3D dostępnym przez wiele źródeł oprogramowania. Współpracujemy również z zespołem projektowym OpenSolaris.

Za jeden z głównych wysiłków podjętych i prowadzonych przez społeczność, uważamy pracę nad wersją LiveCD środowiska LG3D, która dostarczy prostą i skuteczną metodę wypróbowania LG3D bez potrzeby instalowania jakichkolwiek elementów w systemie.

Zakładamy wydanie LG3D w wersji 1.0 przed końcem roku 2006 wraz z funcjonalnością i stabilnością wymaganą w trakcie codziennej pracy.

Jednym z głównych powodów naszego koncentrowania się na prostej instalacji jest zamierzenie systematycznego powiększania społeczności skupionej wokół projektu, niezależnie od stopnia zaawansowania w korzystaniu z komputera. Przebyliśmy bardzo daleką drogę od miejsca, z którego wystartowaliśmy, ale osiągnięcie postawionych na początku celów to nadal odległy punkt na mapie naszej wędrówki z projektem LG3D. Dlatego jesteśmy wdzięczni za przyłączenie sie do projektu i wszelkie wsparcie i pomysły wzbogacające projekt LG3D i odkrywanie wspólnie z nami nowego wymiaru pracy i rozrywki komputerowej.

## 7. Miejsca warte odwiedzenia

- Strona domowa projektu LG3D
  *http://lg3d-core.dev.java.net*
- Oficjalne forum dyskusyjne
  *http://forums.java.net/jive/forum.jspa?forumID=80*
- Przewodniki
  *https://lg3d-core.dev.java.net/tutorial/index.html*
- Inkubator projektów
  *http://lg3d-incubator.dev.java.net/*
- Projekt “LiveCD”
  *http://lg3d-livecd.dev.java.net/*
- Projekt “Art”
  *http://lg3d-art.dev.java.net/*
- Technologie związane z LG3D
  *http://lg3d-core.dev.java.net/lg3d-related-technologies.html*
- Rozważania związane z projektowanym interfejsem użytkownika systemu LG3D
  *https://lg3d-core.dev.java.net/lg3d-ui-design.html*