Etap 1

Programowanie grafiki trójwymiarowej wiąże się z tworzeniem różnorodnych, efektownych dla oka efektów jednak należy pamiętać, że podstawą nawet najbardziej skomplikowanej sceny jest geometria. To baza wyjściowa do nadawania kształtu wizualizacji i ostatecznego „rysu artystycznego".
W ramach pierwszego etapu utworzymy prostą scenę składającą się z obiektów 3D (modeli), podstawowego oświetlenia oraz interfejsu kamery.

Wstęp - Modele (geometria)

Karty graficzne zoptymalizowane są do wyświetlania **trójkątów** jako bazowych prymitywów (spotykane są również linie i punkty, lecz rzadko stosowane z racji swojej dwuwymiarowej natury). Z tego powodu każdy wyświetlany obiekt aproksymowany jest trójkątną siatką. Stosowane są wymiennie angielskie terminy: *mesh*, *triangle mesh*, *polygonal mesh* (*polygonal* – w domyśle triangle). Jeśli konkretne API grafiki trójwymiarowej udostępnia rysowanie innych prymitywów, to na pewno realizowane jest to na niższym poziomie przez sprowadzenie do siatki trójkątów.

Słowem **model** przyjęło się nazywać logiczny obiekt reprezentowany przez siatkę o której mowa wyżej. Modelem może być np. człowiek, krzesło, samochód, zamek. Może istnieć wiele modeli posiadających tę samą reprezentację geometryczną np. dwóch przeciwników w grze różniących się kolorem lub zestawem tekstur.

Siatki dla modeli mogą być przygotowane w różny sposób:

1. **Ręcznie** w kodzie poprzez podanie parametrów wierzchołków. Podstawowym parametrem jest wektor określający pozycję w 3D [x,y,z], ale do zrealizowania konkretnych efektów (w szczególności oświetlenia) często potrzebne są dodatkowe parametry takie jak np. wektor normalny, kolor, jeden lub więcej zestawów współrzędnych tekstury, indeks materiału, wektor styczny/binormalny itp.
2. **Proceduralnie** przy pomocy wzoru na powierzchnię np. latarnia jako bryła obrotowa, tor samochodowy jako rozciągnięty wzdłuż współrzędnej Z wykres funkcji. Szczególnie dużo dostępnych w Internecie źródeł poświęconych jest proceduralnemu generowaniu terenu i powierzchni wody.
3. **Przy pomocy wbudowanych funkcji** do tworzenia prostych kształtów – modele można budować z kul, prostopadłościanów, walców, stożków. W niektórych API dostępne są funkcje tworzące siatki takich obiektów. Jak łatwo zauważyć, podejście sprawdza się tylko dla pewnych obiektów o prostej geometrii.
4. **Wczytane z plików modeli.** To podstawowa metoda umieszczania modeli na scenie w komercyjnych produkcjach takich jak gry komputerowe lub filmy. Artyści modelują modele w specjalistycznych programach np. 3D Studio Max, Maya, Blender, SoftImage i eksportują do plików. Pliki te traktowane są z punktu widzenia programu jak zasoby, często określane mianem **contentu** lub **assetów**.

Uwaga.
W ramach przedmiotu naturalnie nie jest wymagana umiejętność modelowania siatek. Wymagana jest natomiast umiejętność wczytania gotowych modeli z pliku i umieszczenia ich na scenie (szczegóły w definicji zadania). W Internecie dostępne są dziesiątki tysięcy darmowych siatek do pobrania.
Przykładowe formaty:
***.x** – format DirectX. Występuje w odmianach tekstowej i binarnej.
Zaletą jest wbudowana, gotowa do użycia funkcja wczytująca w DirectX. W starszych wersjach DirectX SDK znajduje się dużo modeli w tym formacie.
***.obj** – prosty format zaproponowany przez Wavefront.
http://en.wikipedia.org/wiki/Wavefront_.obj_file

Do powyższych dwóch formatów łatwo jest napisać parser ręcznie jeśli dane środowisko ich natywnie nie wspiera. Do pozostałych sugeruję skorzystać z gotowych, darmowych bibliotek obsługi modeli.
Inne popularne formaty to:
***.3ds, *.max, *.fbx** – Autodesk
***.md2, *.md3** – formaty Quake
***.blend** – format Blendera
COLLADA (*.xml) – w założeniu uniwersalny format wymiany siatek

Biblioteka AssImp napisana w C++ do obsługi różnego rodzaju formatów:
http://assimp.sourceforge.net/

Biblioteka do obsługi formatu 3ds:
http://code.google.com/p/lib3ds/

Przykład wczytywania modeli 3D w XNA:
http://xbox.create.msdn.com/en-US/education/catalog/sample/winforms_series_2

Zadanie

Scena

Tematem przewodnim sceny przestrzeń kosmiczna. Należy umieścić na scenie:

- Obiekt reprezentujący planetoidę – złożoną ze jednej dużej sfery oraz jednej lub kilku połączonych ze sobą półsfer i półwalców na powierzchni planetoidy reprezentujących stację badawczą. Utworzone one powinny być w kodzie przy inicjalizacji programu. Można użyć wszelkich wbudowanych funkcji API graficznego do utworzenia prostych obiektów geometrycznych lub ręcznie podać wierzchołki. Można na powierzchni dodać dodatkowe obiekty reprezentujące np. nierówności powierzchni itp.
- Przynajmniej dwie satelity wokół planetoidy – utworzone dowolną metodą – mogą się składać np. z kilku walców, sfer i/lub prostopadłościanów. Mogą być także wczytane z pliku. Wymaganiem jest, aby dla tych obiektów istniała tylko **jedna siatka**, czyli przed narysowaniem ustawiana jest odpowiednia macierz przekształcenia, która umieści geometrię (fizycznie istniejącą w jednym egzemplarzu) w odpowiednim miejscu na

scenie. Zamiast satelity mogą być inne obiekty o **wspólnej** siatce i **co najmniej dwóch** wystąpieniach (np. łaziki, kosmonauci itp.).

- Co najmniej dwa, dowolne obiekty wczytane z plików. Można skorzystać z plików dołączanych w przykładach do SDK (DirectX, SDK, MDX), pobrać darmowe modele z Internetu (np. http://www.turbosquid.com/) lub ręcznie zamodelować (co jest czasochłonne i odradzane).
  Przykładowe obiekty to np. łazik, satelita, zamodelowane dodatkowe elementy stacji badawczej.

Można zaprojektować też inną scenę o podobnej tematyce, o ile scena posiada podobne założenia: prosty kształt głównego obiektu; istnieją co najmniej dwa obiekty o identycznej geometrii (jak satelity); oraz można umieścić dwa nietrywialne obiekty wczytane z plików.

Oświetlenie

W pierwszym etapie korzystamy z prostego **oświetlenia lokalnego** bazującego na modelu Phonga, które jest wbudowane w starsze wersje dostępnych API. W nowszych wersjach oraz w XNA należy napisać ręcznie shader takiego oświetlenia lub skorzystać z dołączonego przykładu – prawie każdy uwzględnia standardowe oświetlenie. Oświetlenie lokalne szczegółowo jest także omówione na wykładzie.

Na scenie należy umieścić:

- Dwa światła reflektorowe imitujące np. reflektory łazika, satelity lub niezależne reflektory rozstawione wokół stacji.
- Jedno światło kierunkowe imitujące światło pobliskiej gwiazdy.
- Przynajmniej jedno z tych świateł powinno zmieniać kolor (np. światło z satelity imitujące skanowanie powierzchni w różnych długościach fal świetlnych). Można skokowo przechodzić pomiędzy kolorami z ustalonego zestawu kolorów, gładko przechodzić pomiędzy nimi lub zastosować jakąś inną technikę.

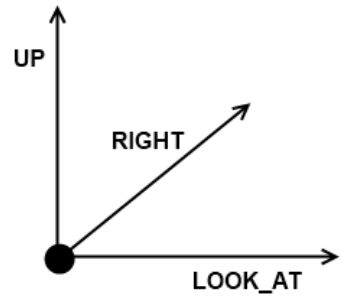
Dla każdego światła należy uwzględnić składnik **diffuse** oraz **specular** z modelu Phonga. Podczas testowania może zaistnieć konieczność zmiany parametrów świateł (kierunku i kąta rozwarcia dla świateł reflektorowych, koloru diffuse), ale nie jest wymagany specjalny interfejs do tego – wystarczy, że autor będzie potrafił sprawnie zmodyfikować odpowiednią wartość w kodzie. Scena w domyśle znajduje się w kosmosie, gdzie ze względu na brak atmosfery, światło nie rozprasza się jednorodnie po scenie. Dlatego składowa **ambient** w modelu Phonga powinna mieć wartość 0.

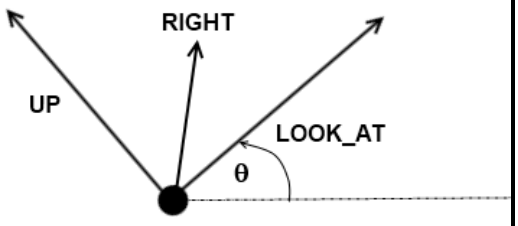
Kamera

DirectX oraz OpenGL nie zawierają w sobie obiektu kamery. To pojęcie abstrakcyjne, wyższego poziomu. Zaprojektowanie wygodnego interfejsu kamery to bardzo ważny element programowania grafiki 3D. Zadaniem kamery jest wykonanie przekształcenia ze współrzędnych świata do współrzędnych kamery a następnie wykonanie rzutowania (zazwyczaj perspektywicznego) do przestrzeni ekranu. Przekształcenia na współrzędnych zapisywane są w

postaci macierzy 4x4 we współrzędnych jednorodnych. Dzięki temu, łatwe i naturalne jest składanie (superpozycja) przekształceń.
Z trójwymiarową kamerą związane są zwykle właściwości widoku takie jak pozycja i kierunek (punkt) patrzenia oraz projekcji takie jak pole widzenia (FOV), stosunek wysokości do długości obrazu, na który przeprowadzane jest rzutowanie a także odległość od bliskiej i dalekiej płaszczyzny obcinania.
Kamera, którą należy zaimplementować w projekcie to kamera umożliwiająca następujące akcje:

- Poruszanie do przodu / w tył
  Przesuwa kamerę wzdłuż aktualnego kierunku patrzenia (w obie strony).
- Poruszanie w lewo / w prawo
  Przesuwa kamerę wzdłuż aktualnego kierunku w lewo (w prawo).
  Jest to kierunek zawsze prostopadły do kierunku „do przodu" oraz „do góry".
- Poruszanie do góry/ w dół
  Przesuwa kamerę wzdłuż aktualnego kierunku „w górę", który pokrywa się z kierunkiem do góry na płaszczyźnie ekranu.
- Spoglądanie góra-dół
  Umożliwia zmianę kierunku patrzenia w osi pionowej. Można tę własność utożsamić z pewnym kątem θ współrzędnych sferycznych zmieniającym się w zakresie $[-\pi/2, +\pi/2]$. Zmiana kąta patrzenia nie powoduje zmiany położenia kamery, wpływa natomiast na bieżące kierunki do przodu (do tyłu) i w górę (w dół).
- Rozglądanie się na boki
  Umożliwia zmianę kierunku patrzenia poprzez obracanie się wokół bieżącego kierunku „w górę". Nie wpływa na położenie kamery, ale modyfikuje kierunki do przodu (do tyłu) oraz w lewo (w prawo). Można tę własność utożsamić z pewnym kątem φ współrzędnych sferycznych z zakresu $[0,2\pi]$, ale uwaga – kąty (θ,φ) w tym zadaniu nie tworzą klasycznego układu współrzędnych sferycznych, ponieważ obrót na boki dokonywany jest płaszczyźnie wyznaczonej przez kierunek patrzenia (a więc wraz ze zmianą kąta θ zmienia się nachylenie płaszczyzny w której realizowany jest obrót o φ).
  Osie, wokół których dokonywane są obroty są zawsze wzajemnie prostopadłe:





Spojrzenie 'w górę' - kierunek patrzenia (LOOK_AT) jest obrócony względem osi wyznaczanej przez kierunek w bok (RIGHT). Kierunek w górę (UP) uaktualnia się, tak aby pozostał prostopadły do pozostałych kierunków.

Dla każdego położenia kamery istnieje lokalnie przyłożony prostokątny układ współrzędnych rozpięty na wektorach (UP, RIGHT, LOOK_AT).

- Obrót kamery zgodnie/przeciwnie do ruchu wskazówek zegara wokół aktualnego kierunku „w przód”. Nie wpływa na położenie kamery, ale modyfikuje kierunki do w górę (w dół) oraz w lewo (w prawo).

Poruszanie się i rozglądanie powinno być płynne i mieć stałą prędkość - dopóki poruszana jest myszka lub wciśnięty klawisz, dopóty akcja się wykonuje. Innymi słowy, aby dla przykładu poruszać się do przodu wystarczy wcisnąć jeden przycisk i przytrzymać – dopóki jest wciśnięty, kamera będzie się poruszać proporcjonalnie do czasu wciśnięcia. Rozglądanie się na boki powinno być nieograniczone – tzn. wciskając jeden przycisk lub przesuwając mysz, możemy wykonywać wiele pełnych obrotów.
Stałą prędkość można zagwarantować przez uwzględnienie czasu w obliczeniach. W ten sposób szybkość poruszania kamery będzie niezależna od szybkości komputera.

Do zainicjowania okna, obsługi urządzeń myszki/klawiatury, wczytywania zasobów należy posłużyć się dowolnym, możliwie prostym i najlepiej gotowym rozwiązaniem.
W przypadku XNA sprawa jest uproszczona, bo uwzględnia obsługę urządzeń wejścia i nie trzeba odwoływać się bezpośrednio do WinAPI.
W DirectX i MDX można rozpocząć od modyfikacji czystego projektu z SDK lub dodawać stopniowo funkcjonalność do jednego z wielu dostępnych w sieci tutoriali.

Można wykorzystać bibliotekę np. DXUT w DirectX (wspomagającą tworzenie urządzenia, obsługę pętli renderowania). Pusty projekt wykorzystujący DXUT nazywa się w SDK „**EmptyProject sample**”

W OpenGL można wykorzystać np. biblioteki GLUT lub SDL.
http://www.opengl.org/resources/libraries/glut/ lub http://www.libsdl.org/

Podsumowanie

Należy utworzyć projekt w wybranym API grafiki trójwymiarowej spośród XNA, OpenGL, DirectX oraz Managed DirectX.
Zadanie składa się z przygotowania:
- Trójwymiarowej sceny złożonej z co najmniej pięciu obiektów: planetoidy, stacji badawczej, dwóch satelit/kosmonautów/łazików posiadających tę samą geometrię oraz dwóch (lub więcej) modeli wczytanych z plików,
- 2 lub więcej świateł reflektorowych oraz 1 światła kierunkowego,
- Interfejsu kamery.

Szczegółowe wymagania poszczególnych elementów opisane są w powyższym dokumencie.