Interakcja Człowiek-Komputer
Modelowanie postaci i otoczenia

Dr inż. Sławomir Samolej

http://ssamolej.prz-rzeszow.pl

Wojny bibliotek interaktywnej grafiki 3D

OpenGL vs. Direct3D

– Coraz trudniej je odróżnić.
– Nigdy próba konwersji z jednej do drugiej nie da „przyspieszenia” aplikacji

Literatura

- http://nehe.gamedev.net – prosty silnik 3D
- http://www.ogre3d.org/ - zaawansowany silnik 3D
- http://www.panda3d.org/ - silnik 3D
- http://www.blender.org/ - program do tworzenia obiektów 3D
- Kevin Hawkins and Dave Astle, *OpenGL Game Programming*
- K. Kuklo, J. Kolmaga, *Blender Kompendium*, Helion 2007
- S. Zerbst, O. Düvel ,*Three-D game engine programming,* Course Technology, Inc. 2004
- D. Shreiner, M. Woo, J. Neider, T. Davis, *OpenGL Programming Guide, Sixth Edition*, Pearson Education, Inc., 2008
- R. S. Wright, Jr., B. Lipchak, OpenGL® SUPERBIBLE, Fourth Edition, Addison Wesley 2007.

Czym dysponujemy?

- Biblioteki graficzne (OpenGL/DirectX)
- Programy do interaktywnej generacji siatek i animacji szkieletowych (3D Studio Max, Blender, Maya, Lightwave, Cinema4D)
- Silniki Graficzne (Darmowe: Pand3D, Ogre3D, XNA, DarkGDK, Jmonkey)

Biblioteki graficzne

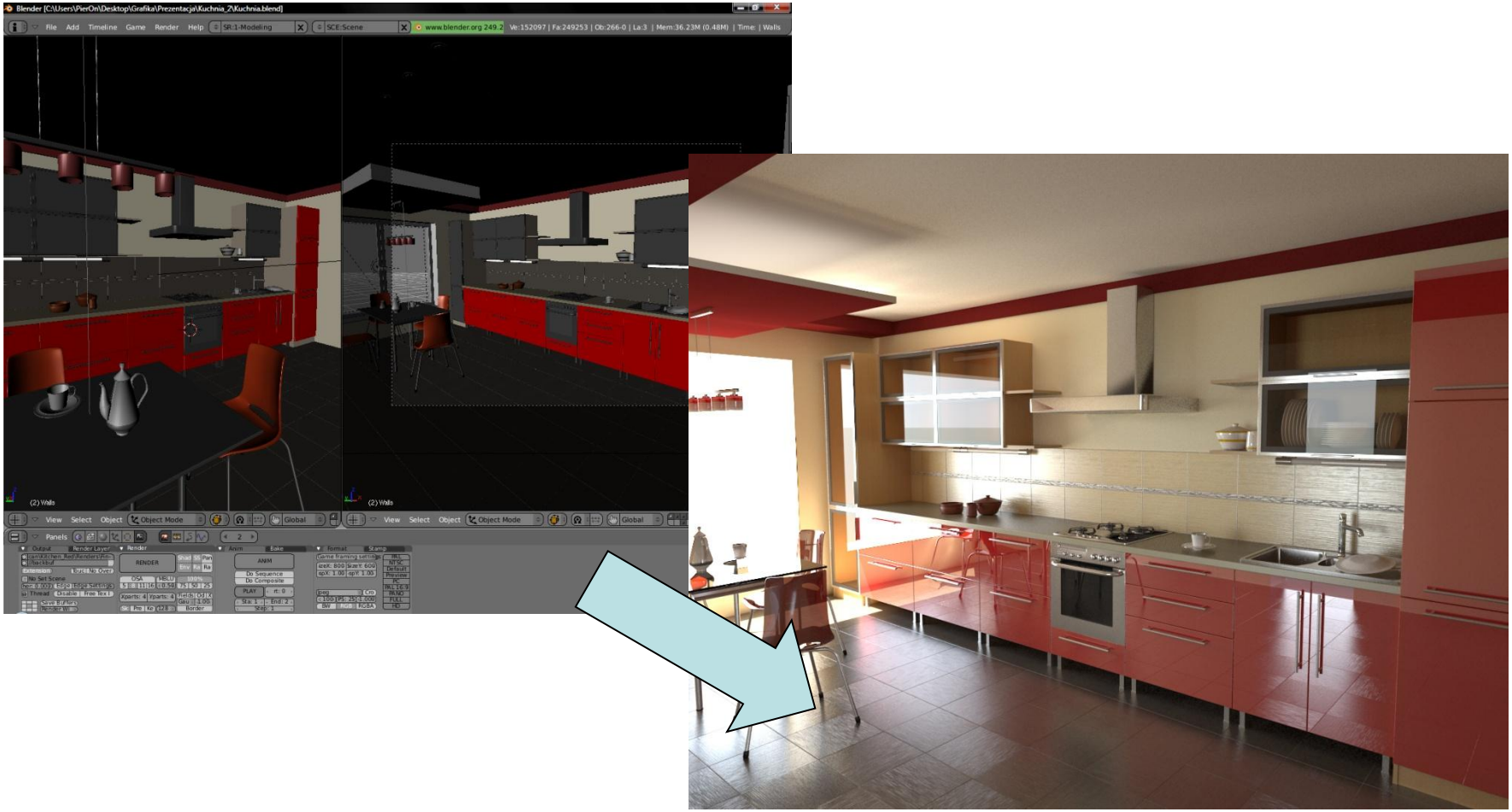
- Udostępniają tzw. prymitywy graficzne, z których można tworzyć siatki
- Pozwalają na cieniowanie, teksturowanie
- Udostępniają transformacje przestrzenne
- Umożliwiają przechowywanie współrzędnych wierzchołków i tekstur
- Umożliwiają definiowanie shaderów

Programy do interaktywnej generacji siatek i animacji szkieletowych (1)

- W założeniu służą do:
  - Generacji wygenerowanych sztucznie zdjęć
  - Tworzenia filmów

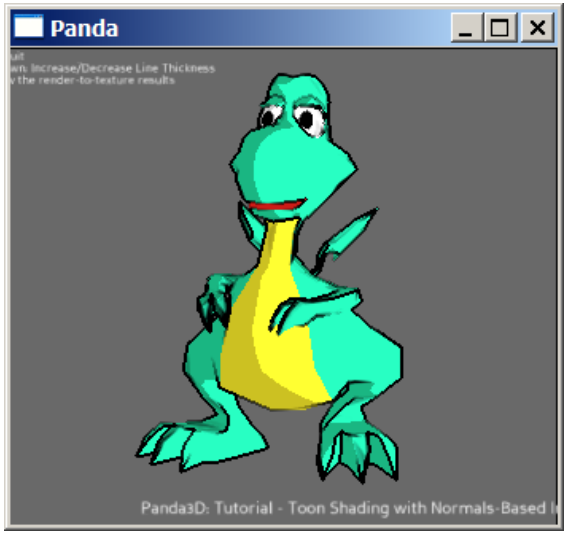
Programy do interaktywnej generacji siatek i animacji szkieletowych (2)

- Przykład modelu przed i po renderingu



Silniki graficzne / Silniki do gier

- Są w stanie wczytać pliki graficzne wygenerowane przez programy do generacji siatek i animacji szkieletowych i efektywnie nimi zarządzać
- Służą do szybszego tworzenia gier oraz interaktywnych aplikacji graficznych

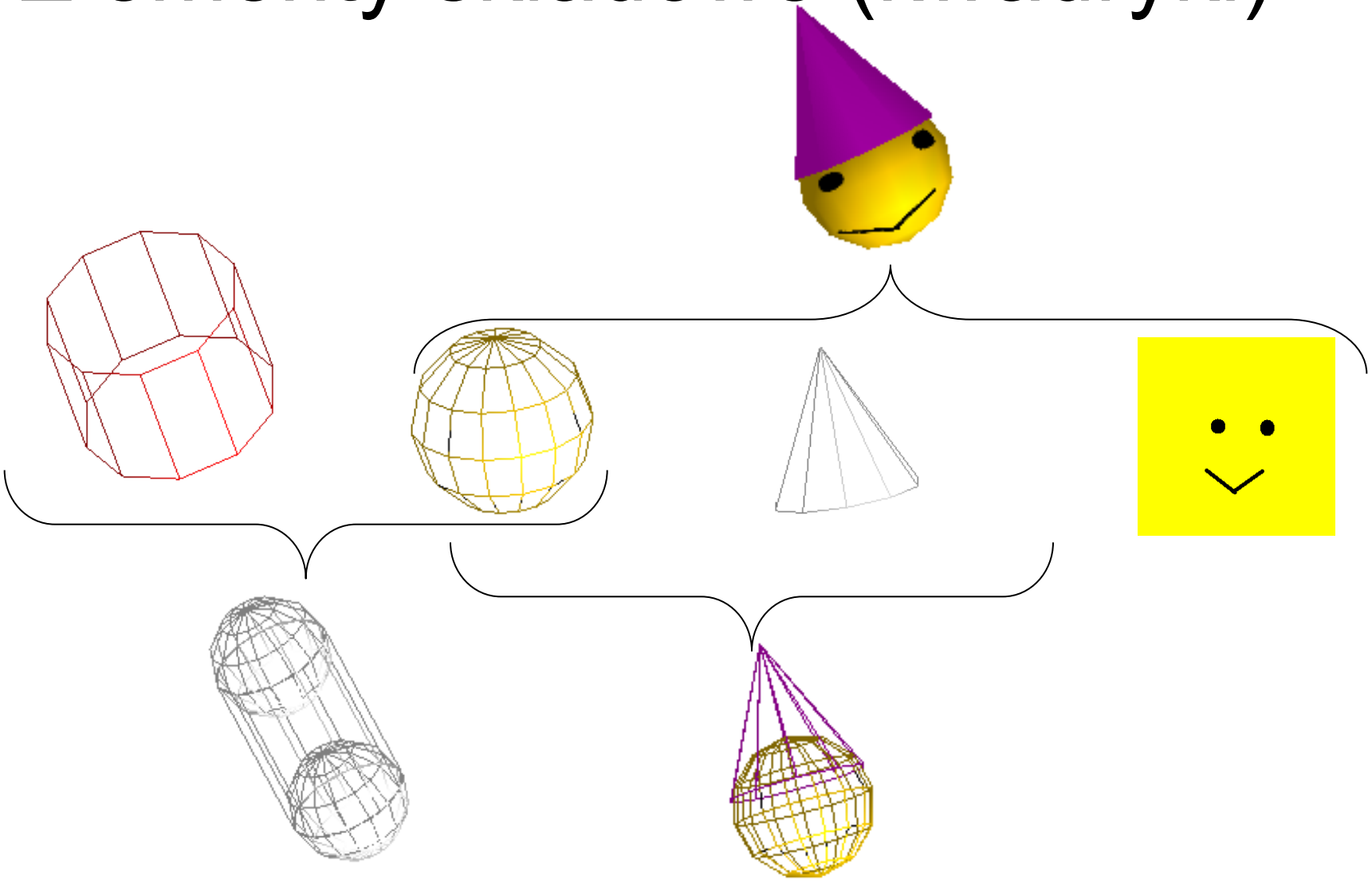


Cel wykładu

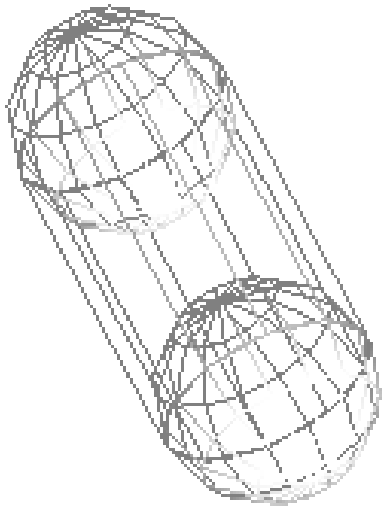
- Przegląd technik tworzenia postaci oraz ich otoczenia na potrzeby interaktywnych aplikacji graficznych.

Podejście 1
Modelowanie prostej postaci z zastosowaniem OpenGL

Elementy składowe (kwadryki)

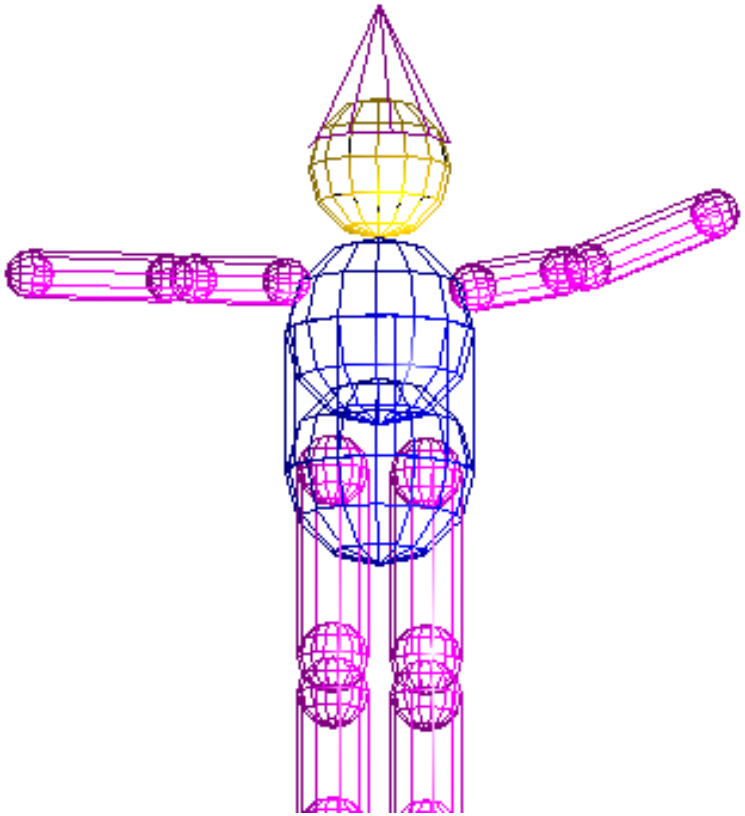


Podstawowy „moduł”



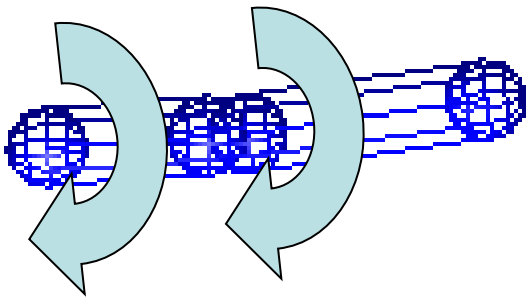
```
void Modul(double dlugosc,
                double srednica
               )
{
     static GLUquadricObj *w1;
     w1=gluNewQuadric();
     glPushMatrix();
     gluCylinder(w1,
                 srednica,
                 srednica,
                 dlugosc,
                 10,
                 1
                );
     gluSphere(w1,srednica,15,6);
     glTranslated(0.0,0.0,dlugosc);
     gluSphere(w1,srednica,15,6);
   glPopMatrix();
}
```

Siatka



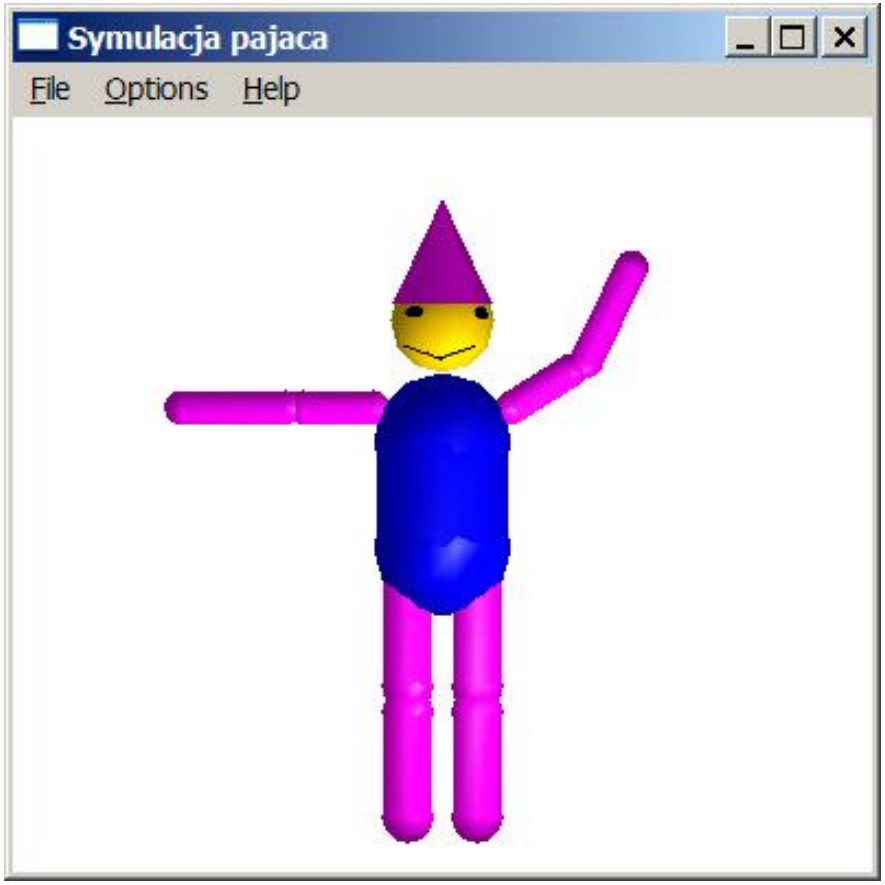
```
void Pajac(double glowa_y,
           double glowa_z,
           double p_ramie_z,
           double p_ramie_x,
           double p_przedramie_z,
           double l_ramie_z,
           double l_ramie_x,
           double l_przedramie_z,
           double sklon,
           double p_udo_x,
           double p_udo_z,
           double p_kolano_x,
           double l_udo_x,
           double l_udo_z,
           double l_kolano_x,
           double p_stopa_x,
           double l_stopa_x
           )
```

Ruch ręki



```
glPushMatrix();
    glTranslated(
            SREDNICA_TULOWIA,
             0,
            DLUGOSC_TULOWIA
            +SREDNICA_TULOWIA/2.0);
    glRotated(
            l_ramie_z+90.0,0,1,0);
    Ramie();
    glTranslated(
            0.,
            0.,
            DLUGOSC_RAMIENIA
            +SREDNICA_RAMIENIA);
    glRotated(l_przedramie_z,0,1,0);
    Przedramie();
glPopMatrix();
```

Prosta animacja



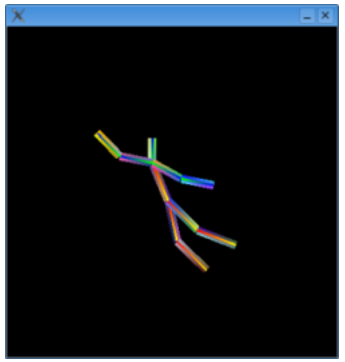

```
case WM_TIMER:
      if(kierunek==0)
      {  d1+=2.0;
         d3+=2.0;
      }
      else
      {  d1-=2.0;
         d3-=2.0;
      }

      licznik++;
      if(licznik > 15)
      { kierunek=(kierunek==1)?0:1;
         licznik=0;
      }
      InvalidateRect(hWnd,NULL,
                              FALSE);
      break;
```

Co dalej?

- Tworzenie bardziej zaawansowanych postaci wymagałoby:
  - Chmury punktów w przestrzeni
  - Powiązania wierzchołów w ciągi np. trójkątów
  - Określenia normalnych dla punktów
  - Wyznaczenia algorytmów modyfikacji siatki, szczególnie w „stawach”
  - Powiązanie współrzędnych tekstur z siatką
  - Opisanie kompleksowej konfiguracji modelu w postaci wektora liczb oznaczających obroty w poszczególnych „stawach”
  - Przygotowanie „ścieżki animacji” – sekwencji wektorów liczb w poszczególnych chwilach czasu, lub wskazanie klatek kluczowych i zastosowanie algorytmów aproksymacji kołowej na ruch pomiędzy klatkami kluczowymi
  - …

Szkic rozwiązania można znaleźć:
http://content.gpwiki.org/index.php/OpenGL:Tutorials:Basic_Bones_System

Zastosowano tam bibliotekę SDL (Simple DirectMedia Layer)
http://www.libsdl.org



Ważna uwaga

- Przedstawione podejście pozwala na łatwe animowanie postaci na podstawie dostarczanego na bieżąco strumienia danych z konfiguracją postaci

Podejście 2
„Ręczne” włączanie modeli postaci w program DirectX/OpenGL

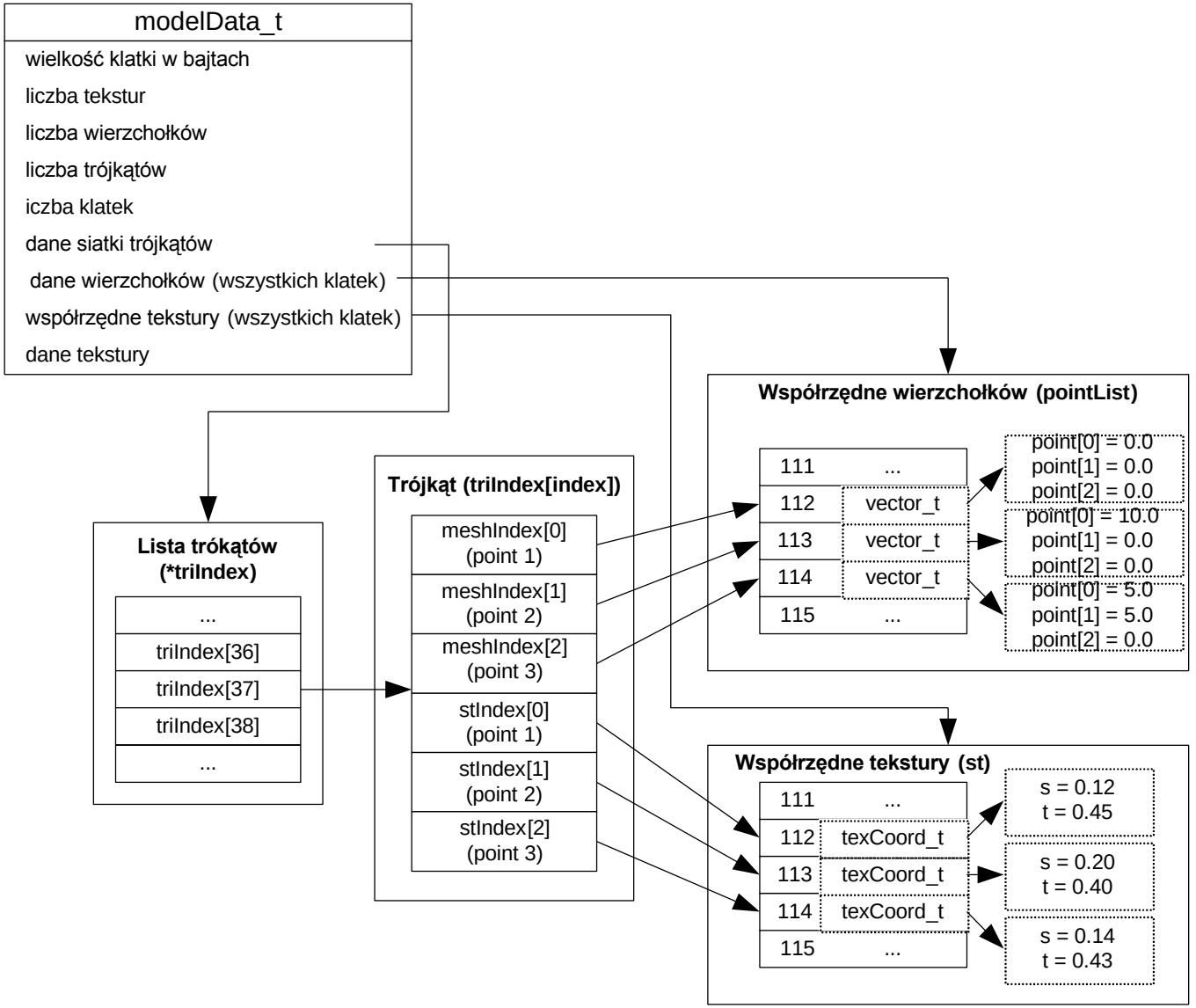
Pomysł

- Istnieją dokumentacje formatu zapisu plików do animacji postaci opracowane przez twórców silników do gier
- Postaci generowane są za pomocą osobnych programów graficznych (3D Studio Max, Blender , Cinema4D, Lightwave itp.)
- Można opracować odpowiednie struktury danych do „wczytania” plików w danym formacie i zastosować jedną z typowych bibliotek graficznych do wyświetlania
- Przykłady takiego podejścia można znaleźć na stronie: www.gametutorials.com

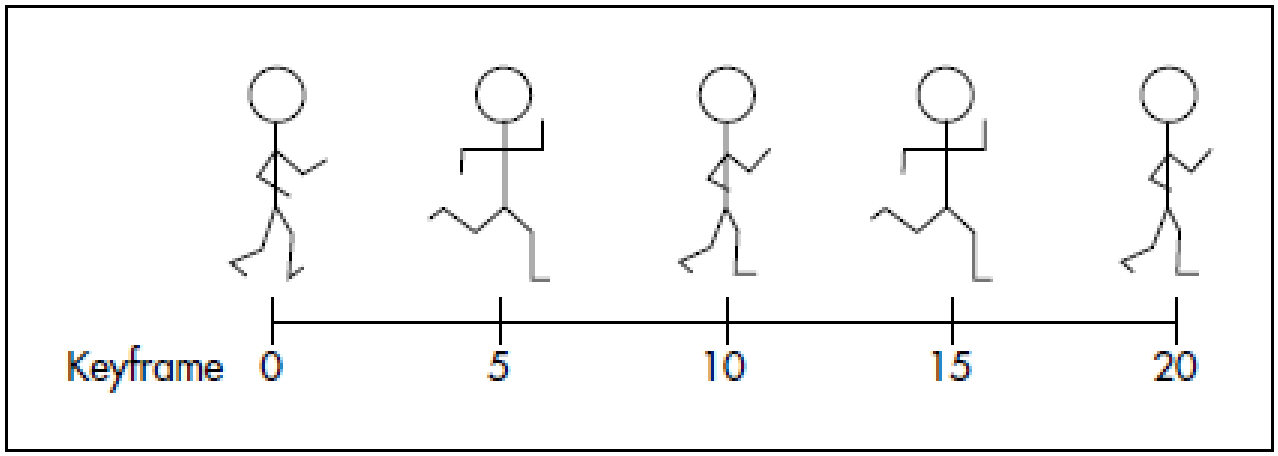
Wybrane formaty plików graficznych do reprezentacji postaci – MD2 - nagłowek

```
typedef struct
{
   int ident;              // identyfikator formatu MD2: "IDP2"
   int version;            // obecnie wersja 8
   int skinwidth;          // szerokość tekstury
   int skinheight;         // wysokość tekstury
   int framesize;          // rozmiar klatki (w bajtach)
   int numSkins;           // liczba tekstur
   int numXYZ;             // liczba wierzchołków
   int numST;              // liczba współrzędnych tekstury
   int numTris;            // liczba trójkątów
   int numGLcmds;          // liczba komend OpenGL
   int numFrames;          // całkowita liczba klatek
   int offsetSkins;        //położenie (w pliku) nazw tekstur(każdy po 64 bajty)
   int offsetST;           // położenie (w pliku) współrzędnych tekstury
   int offsetTris;         // położenie (w pliku) siatki trójkątów
   int offsetFrames;       // położenie (w pliku) danych klatki (wierzchołków)
   int offsetGLcmds;       // typ używanych poleceń OpenGL
   int offsetEnd;          // koniec pliku
} modelHeader_t;
```

Wybrane formaty plików graficznych do reprezentacji postaci MD2 – interpretacja danych



Wybrane formaty plików graficznych do reprezentacji postaci - klatki kluczowe



Wybrane formaty plików graficznych do reprezentacji postaci MD3 - komentarz

- MD3
  - Format MD3 jest również dziełem firmy id Software i stanowi rozwinięcie formatu MD2. Format ten powstał na potrzeby gry Quake3. W przeciwieństwie do plików .md2 postać podzielona jest na 3 oddzielne pliki: głowa.md3, część_górna.md3 i część_dolna.md3. Jeśli postać posiada broń to występuje oddzielny plik broń.md3. Każda z części ciała jest połączona za pomocą tak zwanych "tag"-ów (uchwytów). Każda z postaci zapisana w formacie MD3 posiada trzy uchwyty. I tak za pomocą jednego z uchwytów model głowy połączony jest z górną częścią ciała. Drugi z uchwytów łączy górną część ciała z nogami. Trzeci z uchwytów jest pomocny w umocowaniu broni do prawej ręki postaci. Poszczególne części ciała mają swoje własne listy animacji. Przykładowo, gdy postać wywróci się do tyłu obie części będą przewracać się razem, ponieważ górna część ciała jest przymocowana do nóg. Podobnie stanie się z głową, ponieważ jest przymocowana do górnej części ciała. Korzyścią takiego rozwiązania jest posiadanie niezależnej animacji dla każdej z części ciała. Oznacza to, że postać może wykonywać animację strzelania niezależnie od tego, co robią nogi może ona skakać lub lądować. W ten sposób nie trzeba robić animacji dla każdego przypadku.

Wybrane formaty plików graficznych do reprezentacji postaci

- MD2/MD3

Inne formaty zapisu obiektów graficznych

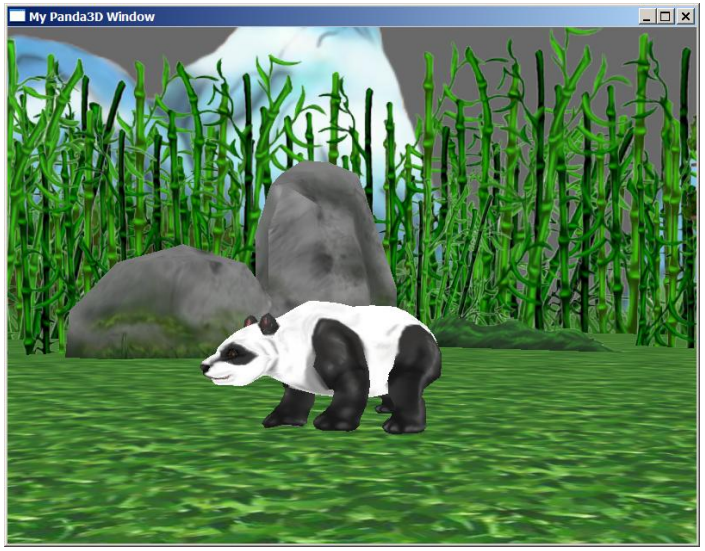
- X – DirectX
- Ogre3D Mesh – Ogre3D
- EGG – Panda3D
- OBJ – 3D Studio Max
- Uwagi:
  - Uzyskanie opisu tych formatów plików jest utrudnione.
  - Do wyświetlania postaci w tych formatach stosuje się inne podejście

Uwaga

- Tutaj jesteśmy w stanie tylko „odgrywać” przygotowane wcześniej animacje

Podejście 3
Zastosowanie silników graficznych do animacji postaci

Pomysł



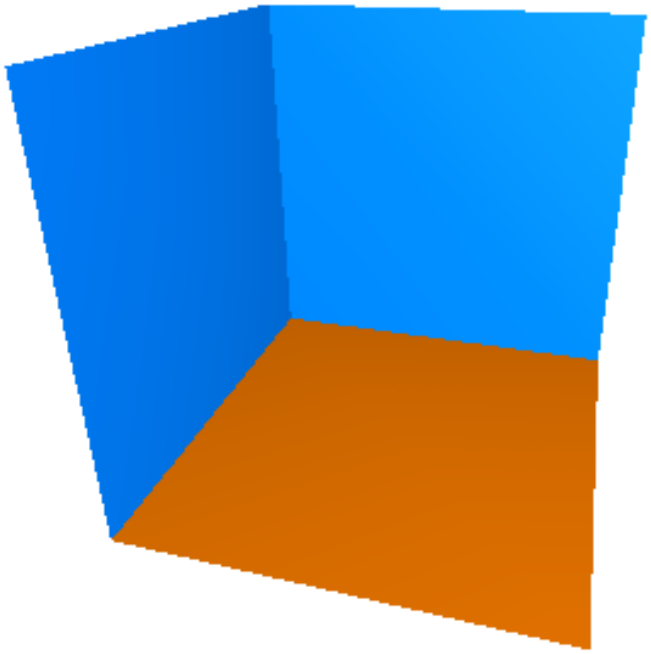
- Dysponujemy animacjami szkieletowymi postaci wygenerowanymi w danym programie graficznym
- Dysponujemy „eksporterami” formatów graficznych do formatów akceptowanych przez dany silnik graficzny
- Wczytujemy gotowy zestaw animacji szkieletowych do silnika

Uwagi dotyczące tworzenia interaktywnych aplikacji 3D

- W zasadzie nie są istotne formaty zapisu danych, tylko, czy istnieje możliwość efektywnego przeniesienia animacji wykonanej w wdanym programie do generacji grafiki na dany silnik graficzny (sprawdzone „exportery”)
- Do tej pory animacje szkieletowe były „odgrywane” w zależności od stanu gry.
- Istnieją takie formaty zapisu, które umożliwiają załadowanie wstępnie przygotowanego pliku z animacją i przejęcie jago kości przez np. system Kinect.

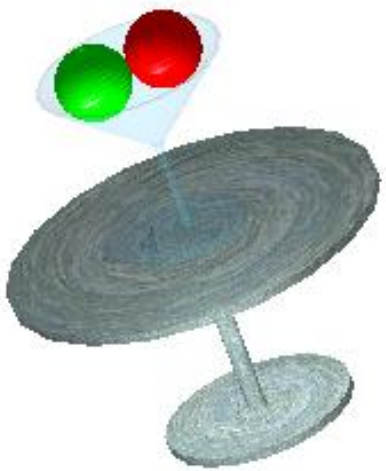
Modelowanie otoczenia z zastosowaniem OpenGL

Proste pomieszczenie



```
glBegin(GL_QUADS);
        glColor3d(1,0.5,0);
        glNormal3d(0,1,0);
        glVertex3d( 20,-20, 20);
        glVertex3d( 20,-20,-20);
        glVertex3d(-20,-20,-20);
        glVertex3d(-20,-20, 20);

        glColor3d(0,0.5,1);
        glNormal3d(0,0,1);
        glVertex3d( 20,-20,-20);
        glVertex3d( 20, 20,-20);
        glVertex3d(-20, 20,-20);
        glVertex3d(-20,-20,-20);

        glColor3d(0,0.5,1);
        glNormal3d(1,0,0);
        glVertex3d(-20,-20,-20);
        glVertex3d(-20, 20,-20);
        glVertex3d(-20, 20, 20);
        glVertex3d(-20,-20, 20);
glEnd();
```

Prosty mebel



- Stół z 9 zateksturowanych kwadryk
- Kieliszek z 9 kwadryk z włączonym mieszaniem kolorów
- Kody – 2 kule

Prosty prostokątny świat

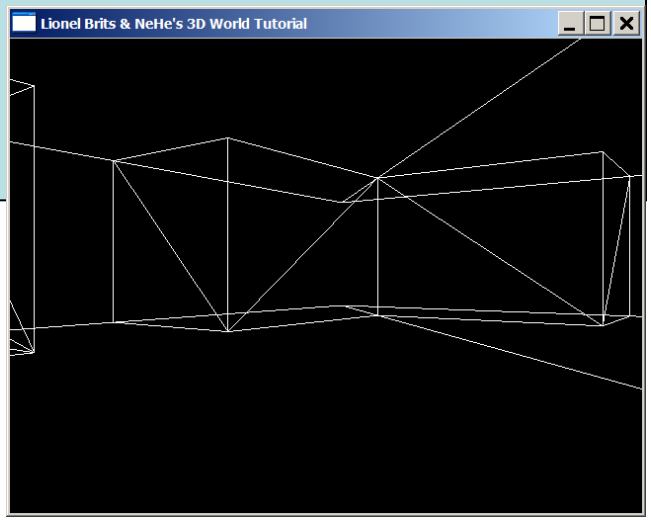
- Struktura zapamiętania wierzchołków:

```
typedef struct tagVERTEX
{float x, y, z;           // 3D Coordinates
float u, v;               // Texture Coordinates
} VERTEX;                 // Call It VERTEX
```

- Skrypt :

```
NUMPOLLIES 36
// Floor 1
-3.0  0.0 -3.0 0.0 6.0
-3.0  0.0  3.0 0.0 0.0
 3.0  0.0  3.0 6.0 0.0

-3.0  0.0 -3.0 0.0 6.0
 3.0  0.0 -3.0 6.0 6.0
 3.0  0.0  3.0 6.0 0.0
```

- Załadowanie skryptu z pliku :



Poruszanie w prostym świecie

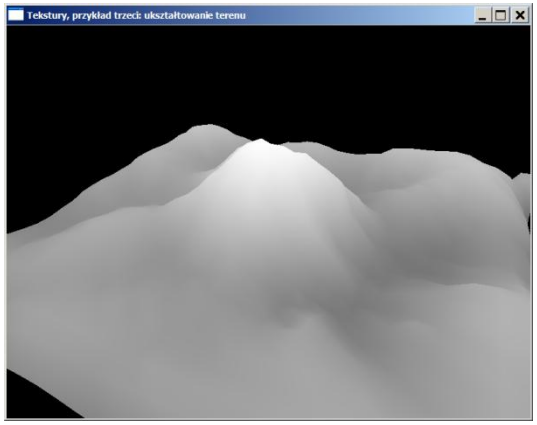

```
// Obsługa klawiszy
if (keys[VK_RIGHT])    // Right Arrow Being Pressed?
{heading -= 1.0f;
 yrot = heading; }     // Rotate To The Left
if (keys[VK_LEFT])     // Left Arrow Being Pressed?
{heading += 1.0f;
 yrot = heading; }     // Rotate To The Right
if (keys[VK_UP])       // Up Arrow Being Pressed?
{ xpos -= (float)sin(heading*piover180) * 0.05f;
  zpos -= (float)cos(heading*piover180) * 0.05f; } //…
if (keys[VK_DOWN])
{ xpos += (float)sin(heading*piover180) * 0.05f;
  zpos += (float)cos(heading*piover180) * 0.05f; } //…
// Wyświetlanie scany
GLfloat xtrans = -xpos;
GLfloat ztrans = -zpos;
GLfloat ytrans = 0 //?
GLfloat sceneroty = 360.0f - yrot;
glRotatef(sceneroty,0,1.0f,0);
glTranslatef(xtrans, ytrans, ztrans);
```

Mapy wysokości - idea

- 2 – wymiarowy plik graficzny może być zastosowany do generowania mapy wysokości
- Kolor danego piksela może być zinterpretowany jako wysokość:

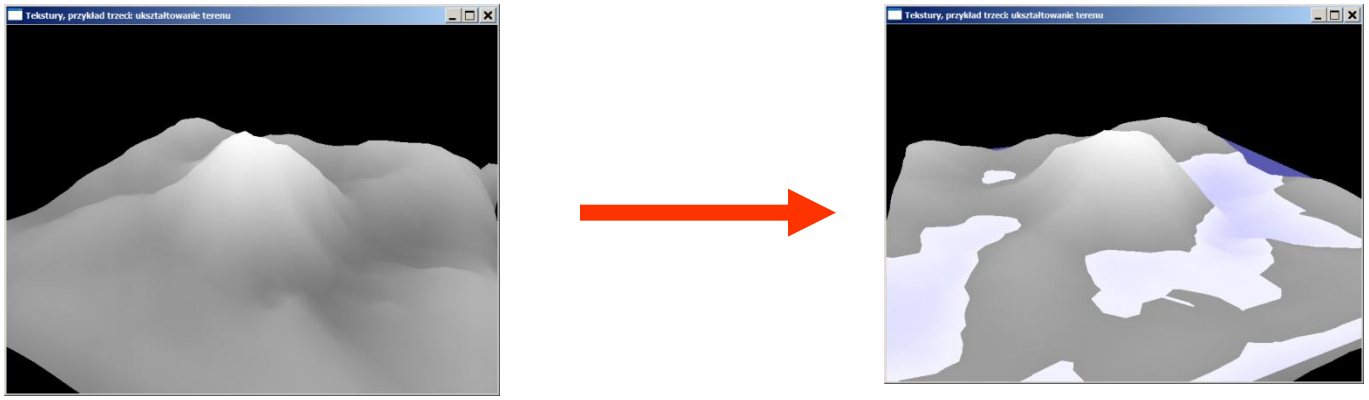
Bitmapa:

Teren:

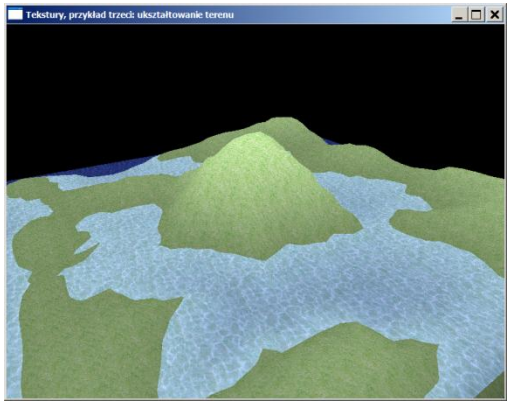


Mapy wysokości – podstawowe triki

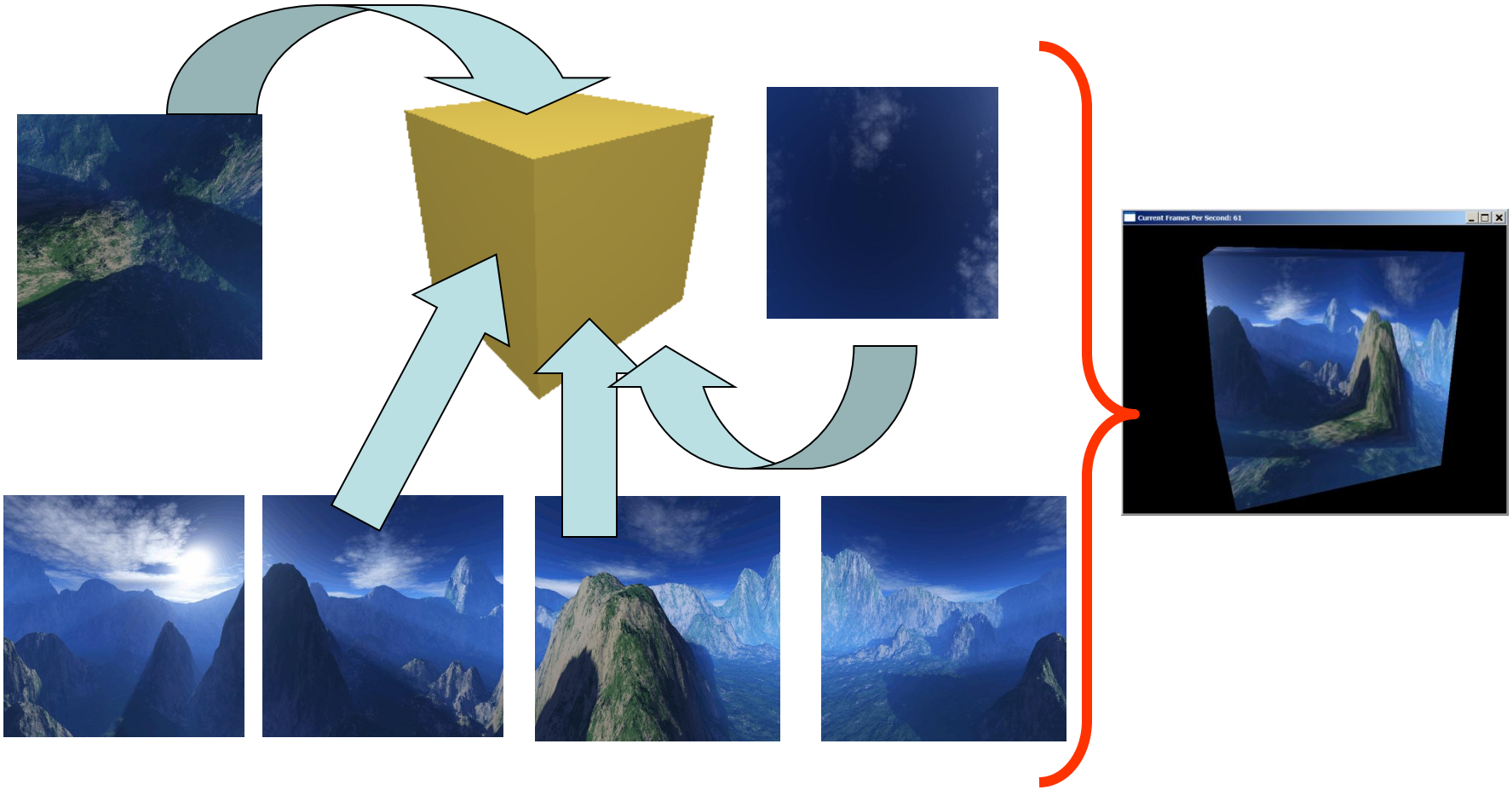
- Falująca woda:



- Teksturowanie:



SKY BOX



Mapy wysokości – dodatkowe triki

- Multiteksturowanie + wolumetryczna mgła + detekcja kolizji

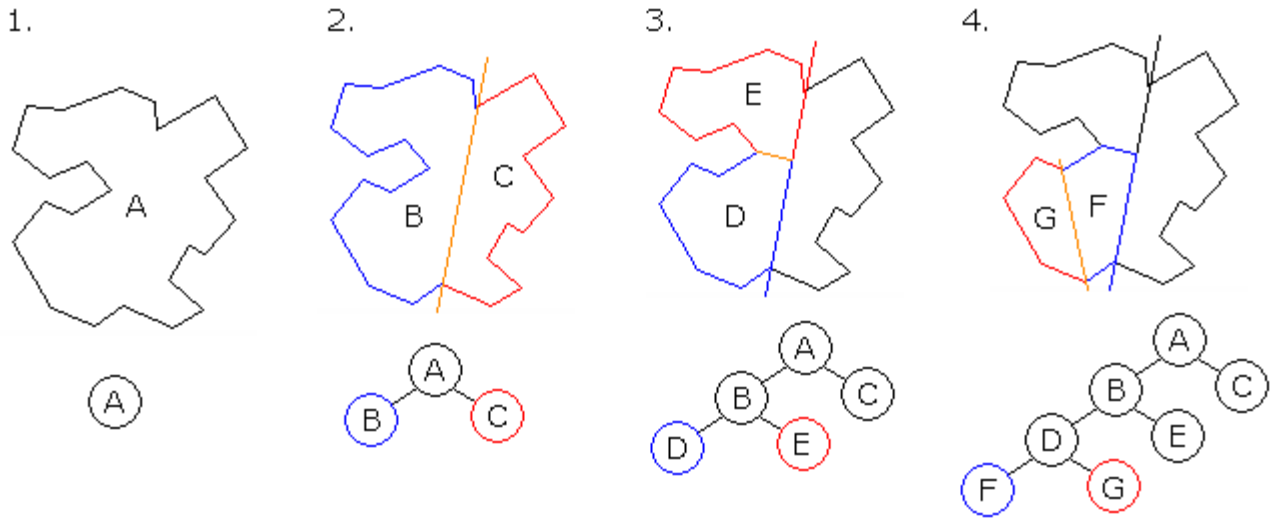
Co dalej?

- Do tworzenia elementów otoczenia można się posłużyć algorytmami (L-drzewa, fraktale)
- Gdy otoczenie ma być interaktywne, należy wprowadzić detekcję kolizji
- Tworzenie złożonych modeli pomieszczeń „na piechotę” wymaga znaczącego nakładu pracy
- Powstaje problem na ile złożone otoczenie jesteśmy w stanie efektywnie płynnie animować w czasie rzeczywistym
- Reprezentacja danych pozostaje tylko w postaci numerycznej – niemożliwej do oceny i modyfikacji przez osobę nie znającą standardu OpenGL.

Drzewa BSP

- **Binary Space Partitioning** jest algorytmem obliczania widoczności na podstawie sortowania obiektów 3D w drzewo binarne. Obliczanie widoczności odbywa się na zasadzie sprawdzania, po której stronie płaszczyzny danego wierzchołka znajduje się kamera (obserwator). Na tej podstawie wybierany jest **lewy** lub **prawy syn** wierzchołka (bardziej odpowiednie jest określenie **przedni** lub **tylny syn**), który sam zawiera swoją płaszczyznę podziału i dwóch synów okerślających obiekty będące z przodu lub z tyłu tej płaszczyzny. Ostatnim synem jest **liść**, który zawiera właściwą geometrię do wyświetlenia. Dzięki temu szybko odrzucana jest znaczna część niewidocznych obiektów.
- Po wybraniu **liścia** często następuje sprawdzanie jakie inne **liście** są z niego widoczne. Najczęściej dzieje się to przy użyciu **Portable Visibility Sets** czyli **pola bitowego**, którego poszczególne bity określają po kolei widoczność każdego liścia. Bit określający widoczność samego siebie jest ma zawsze wartość 1.
- **Zalety i wady BSP**
  - **BSP** jest algorytmem bardzo szybkim i często stosowanym, szczególnie w grach komputerowych. Wadą **BSP** jest nieumiejętny podział otwartego świata 3D, dlatego jest zwykle wykorzystywany dla zamkniętych obszarów. Dla otwartych przestrzeni lepszym rozwiązaniem jest użycie drzewa ósemkowego (octree).

Generacja drzewa BSP



- 1. A jest korzeniem drzewa i całym złożonym obiektem
  2. A jest dzielone na B i C
  3. B jest dzielone na D i E
  4. D jest dzielone na F i G, które są wypukłe, czyli stają się liśćmi

Standard przemysłowy - Quake BSP

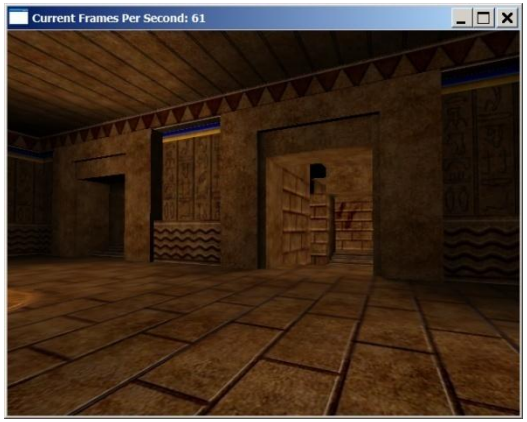
- Zaproponowany przez ID Software (DOOM, QUAKE),
- Stosuje algorytm tworzenia drzew BSP do podziału przestrzeni na wypukłe wielościany
- Przechowuje informację o reometrii otoczenia oraz innych jego składnikach (dodatkowych obiektach, mapach oświetlenia, teksturach, shaderach itp.)
- Nagłówek pliku składa się się z listy „lumps” – kostek z danymi wskazującymi, gdzie w pliku znajdują się poszczególne rodzaje danych.

Elementy nagłówka QUAKE BSP

- Entities – gdzie w przestrzeni są obiekty (wrogowie, bronie), jakie dźwięki odtwarzać, jakie drzwi otwierać,
- Planes – Nieskończone płaszczyzny dzielące przestrzeń na wierzchołki
- Nodes – wierzchołki powstałe przez podział płaszczyznami
- Leaves – liście wypukłe wielokąty
- Visibility – lista potencjalnie widocznych liści
- Textures – lista tekstur powiązanych z poszczególnymi liśćmi
- Faces – parametry wpływające na daną powierzchnię (tekstury, mapy oświetlenia itp.)
- Vertices and Meshes – wspierają proces teksturowania

Quake BSP – praktyczne uwagi

- Pliki Quake BSP można tworzyć z zastosowaniem darmowego oprogramowania Q3Radiant
- Zastosowanie plików Quake PSP umożliwia:
  - Otrzymywanie efektów oświetlenia z zastosowaniem map oświetlenia (multiteksturowanie)
  - Zastosowanie szybkich algorytmów obliczania kolizji
  - Włączanie w strukturę pliku elementów świata
  - Włączanie w strukturę pliku shaderów



Otoczenie w silnikach do gier

- Część z silników akceptuje format Quake BSP
- Pozostałe zakładają możliwość tworzenia elementów otoczenia z (animowanych lub statycznych) obiektów graficznych w ustalonym wewnętrznym formacie
- Do otwartych przestrzeni pozwalają na wprowadzanie map wysokości
- Umożliwiają również zaawansowane zarządzanie modelami otwartej przestrzeni

Przykłady aplikacji

Przykłady aplikacji (1)

Prosta sieciowa gra przygodowa

- Przestrzeń – Quake BSP
- Postaci/przedmioty – MD2
- Muzyka – Direct Sound
- Sieć WinSock2

Przykłady aplikacji (2)

- Wirtualna Katedra Informatyki i Automatyki – Plik Quake BSP + silnik Dark Gdk

- Program odwzorowujący ćwiczenia – Postać wykonana w Blenderze, silnik Ogre3D

Przykłady aplikacji (3)

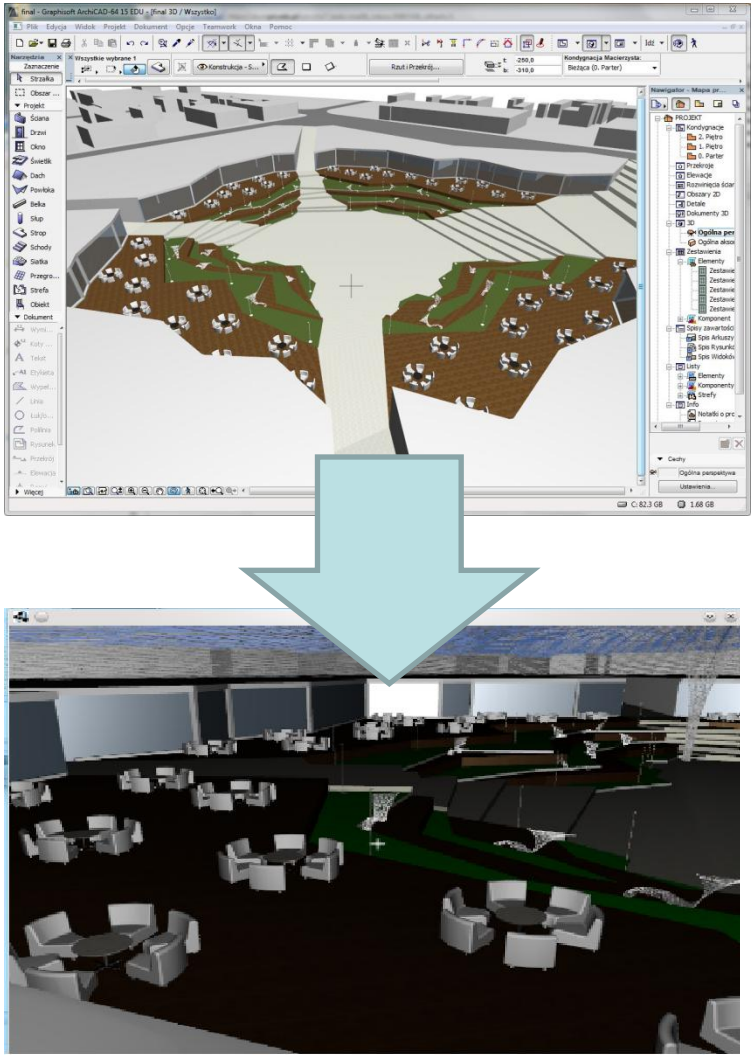
- Program edukacyjny, w którym odgrywane są sceny i dziecko podejmuje decyzje – Animacje opracowane w Blenderze, wczytane do silnika Panda3D

Przykłady aplikacji (4)

- Program naśladujący ruchy dziecka – Postaci wykonane w Blenderze, Aplikacja wykonana w silniku Panda3D, Sprzężenie wizyjne - OpenCV

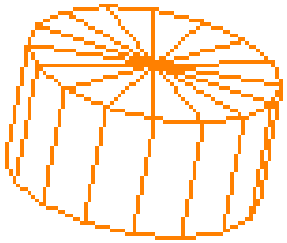
Przykłady aplikacji (5)

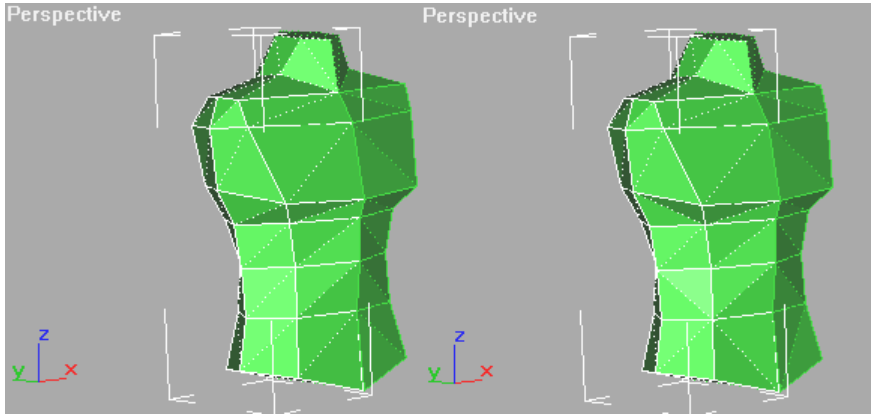
- Interaktywna animacja obiektów architektonicznych – Model ArchiCad, interaktywna animacja – XNA, Ogre3D, JMonkey



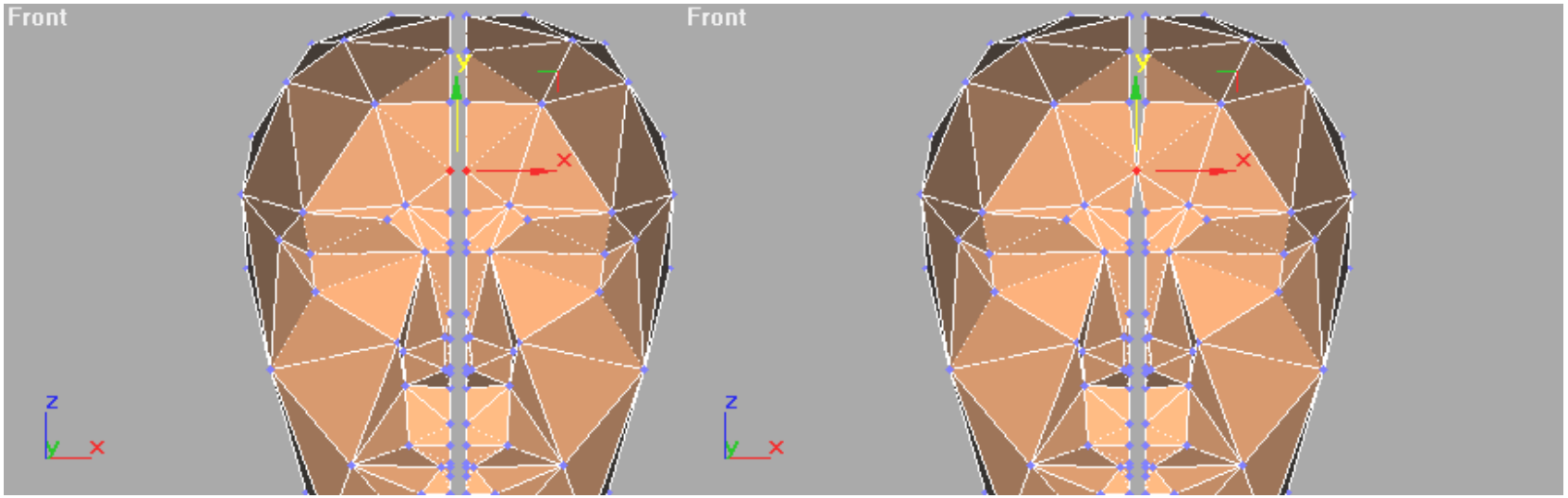
Zasady tworzenia postaci w programach do generacji siatek

1. Modyfikowanie bazowej siatki

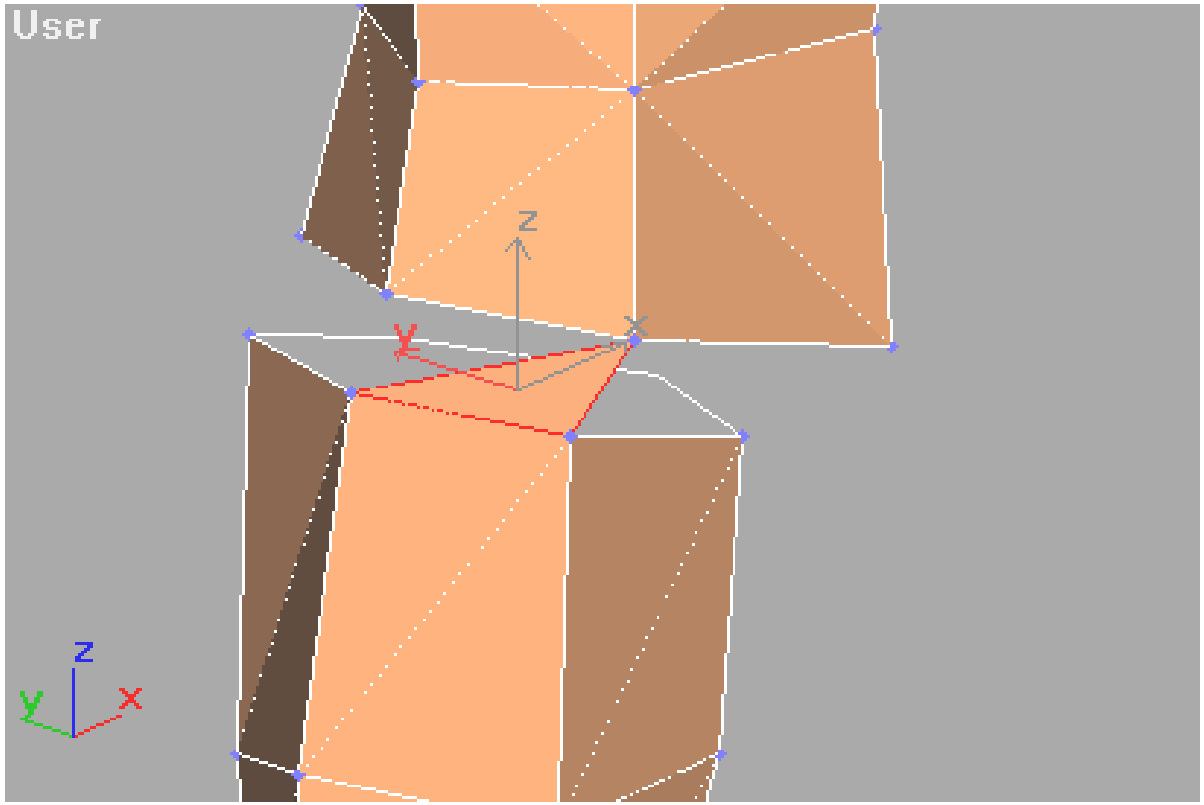




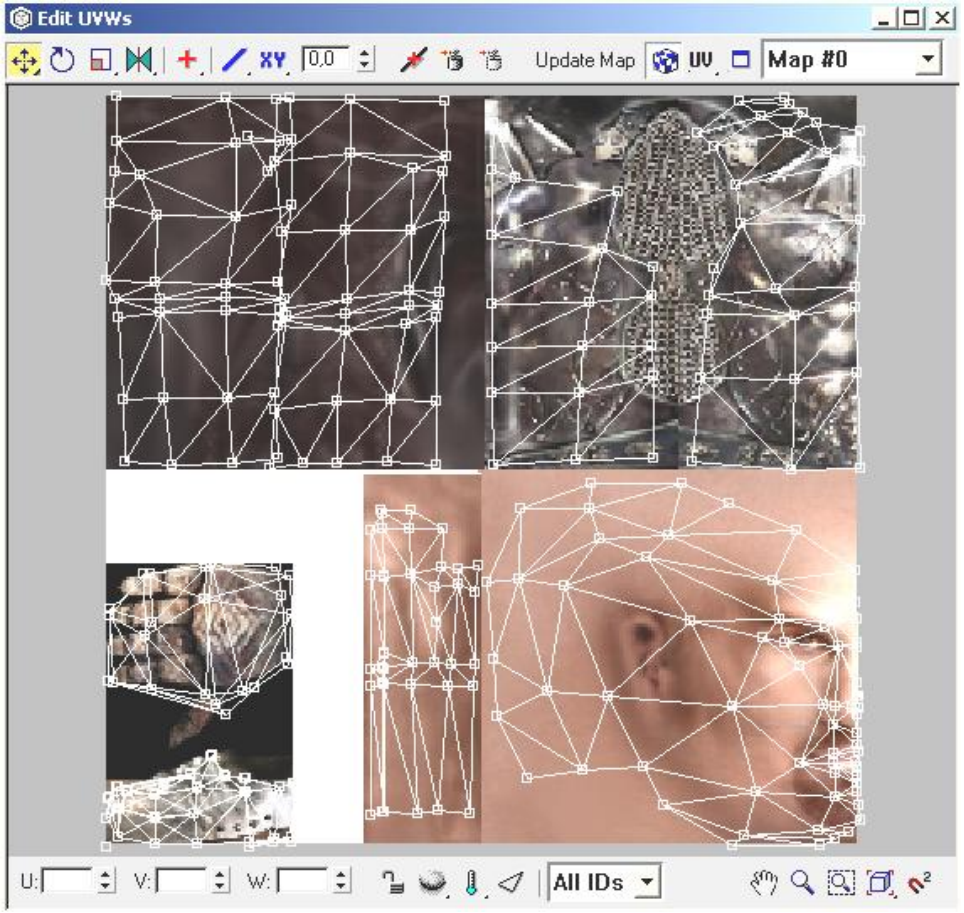
2. Buduje się połowę postaci – drugą otrzymuje się prze „odbicie” i „sklejenie”



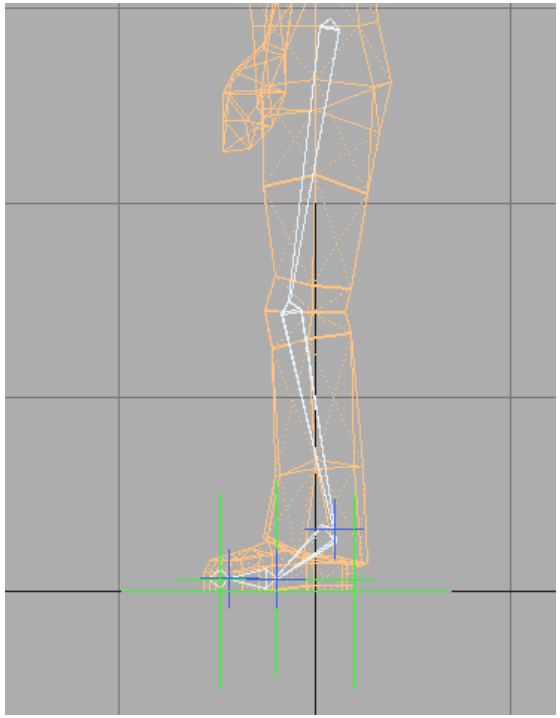
3. Jeśli postać składa się z kilku oddzielnych siatek, trzeba je „skleić”

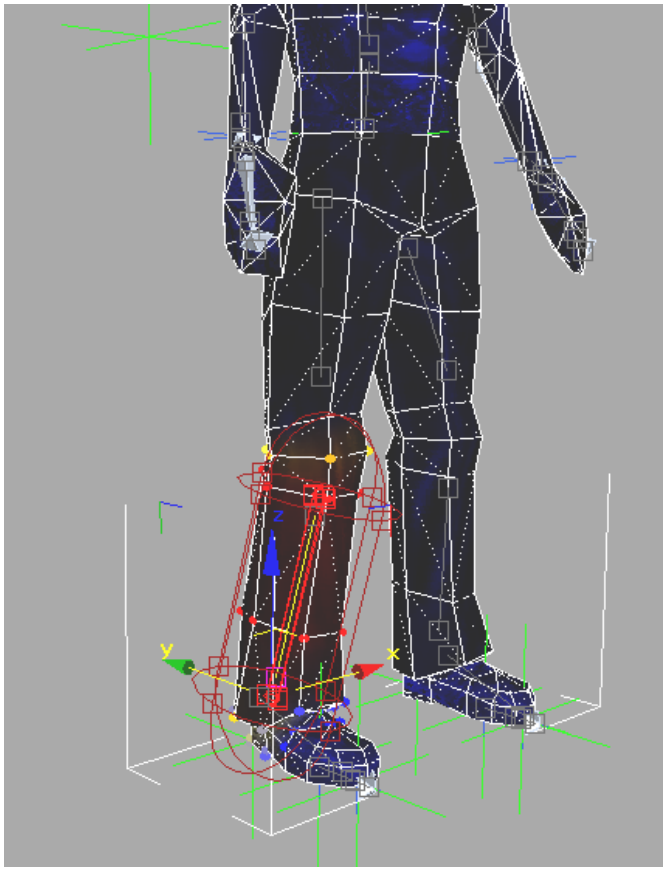


4. Z węzłami siatki trzeba połączyć odpowiednie współrzędne tekstur

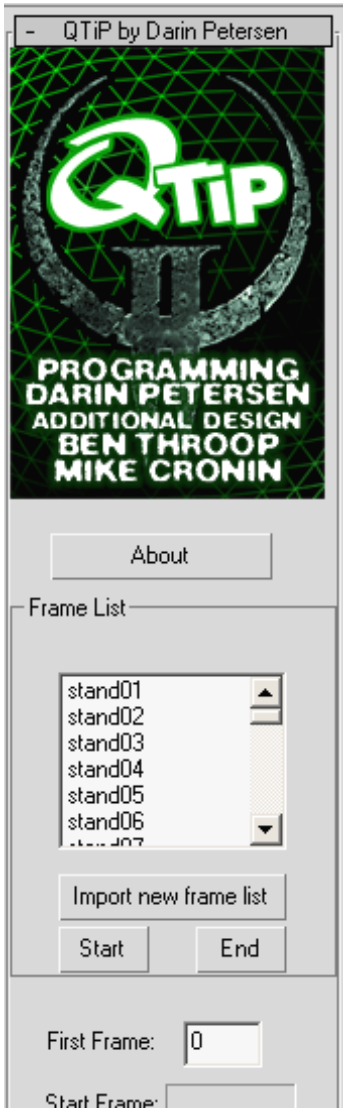


5. Z siatką należy powiązać „kości”; poruszanie kości pozwala na naturalną animację ruchu postaci



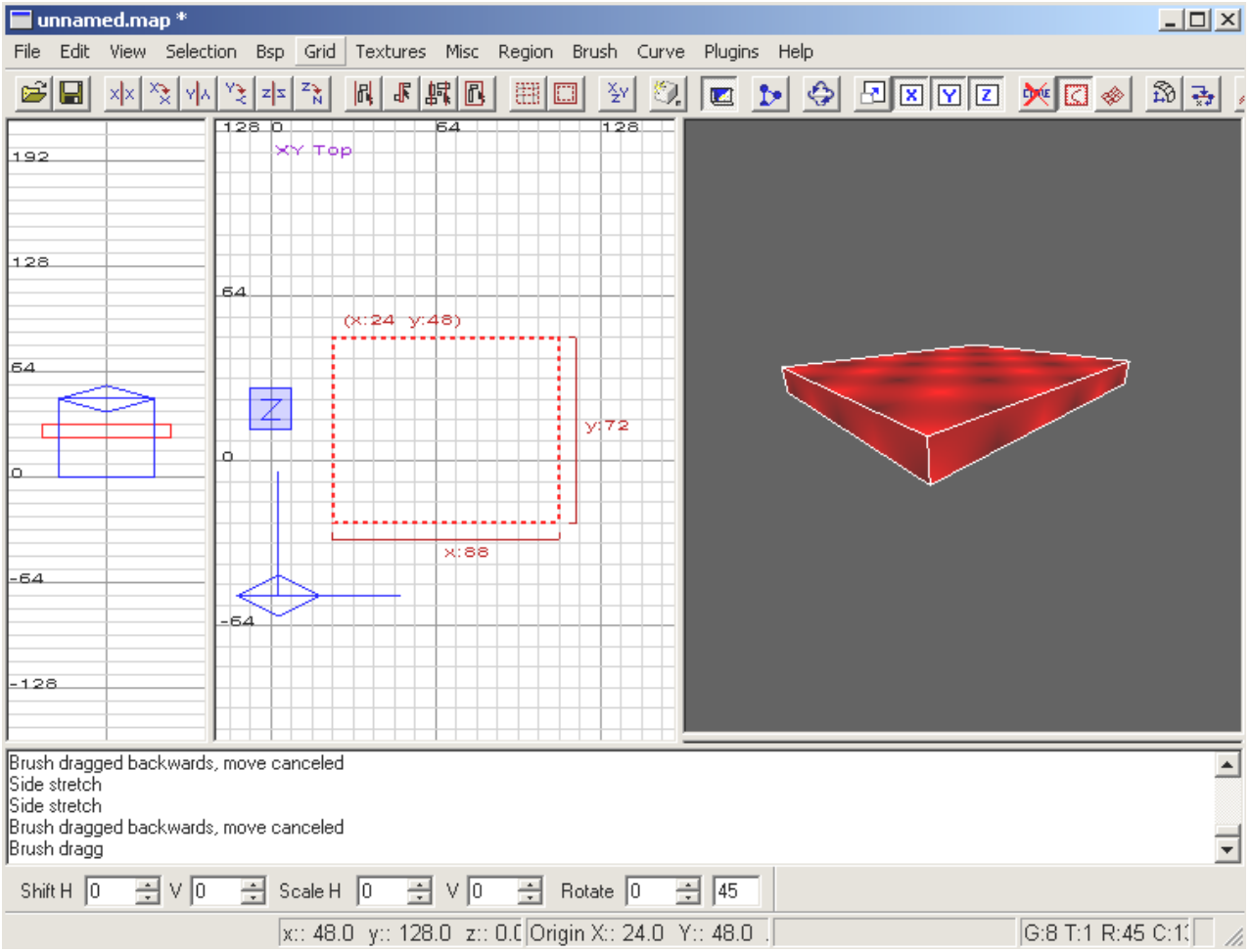


6. Po zdefiniowaniu ruchów w postaci ścieżek animacji dokonuje się eksportu postaci do zadanego formatu graficznego



Zasady budowania modelu otoczenia w Q3Radiant

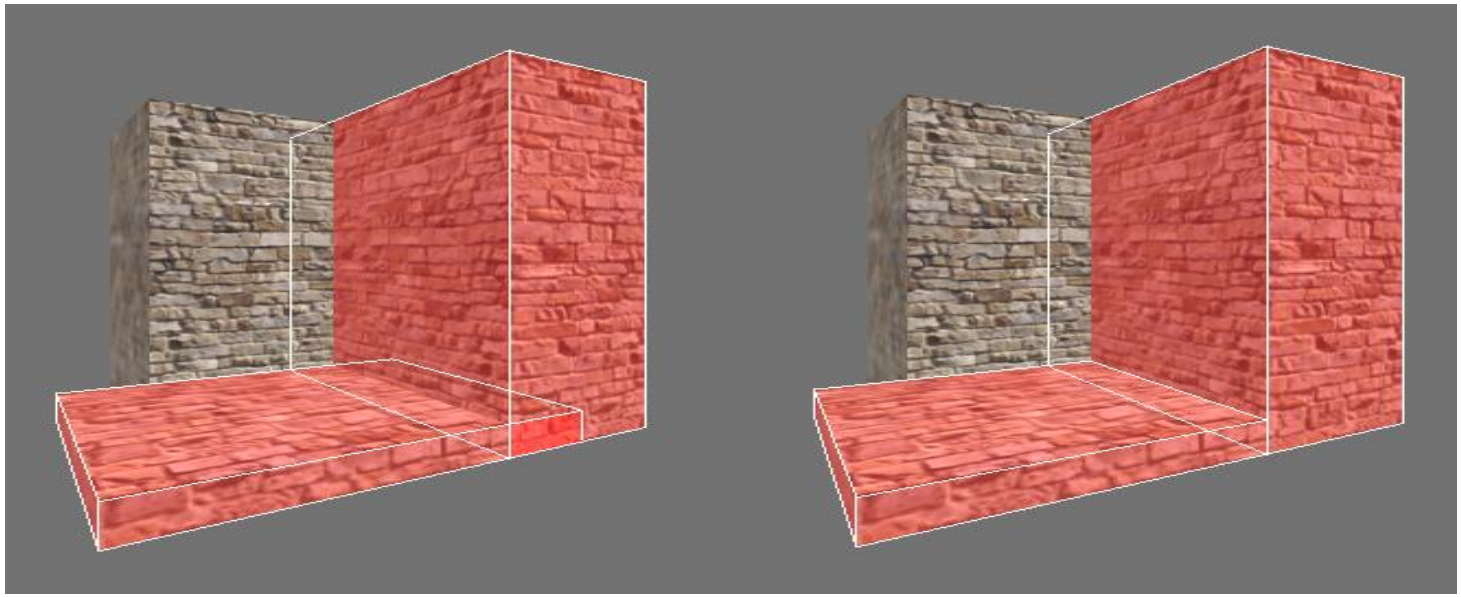
1. Tworzenie modelu zaczyna się od prostopadłościanu



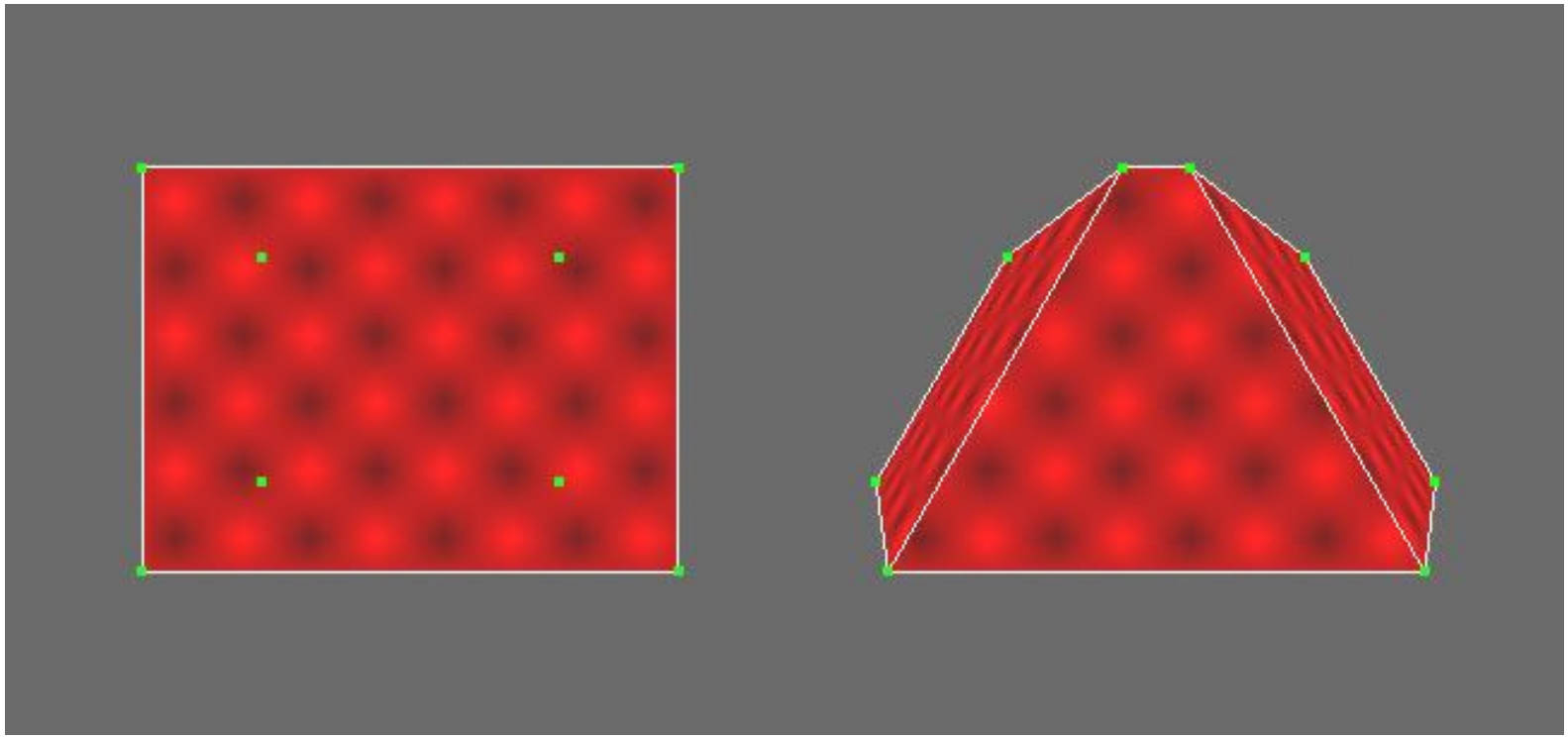
2. Bryły nie mogą na siebie zachodzić, prostopadłościany można skalować i obracać

Źle:

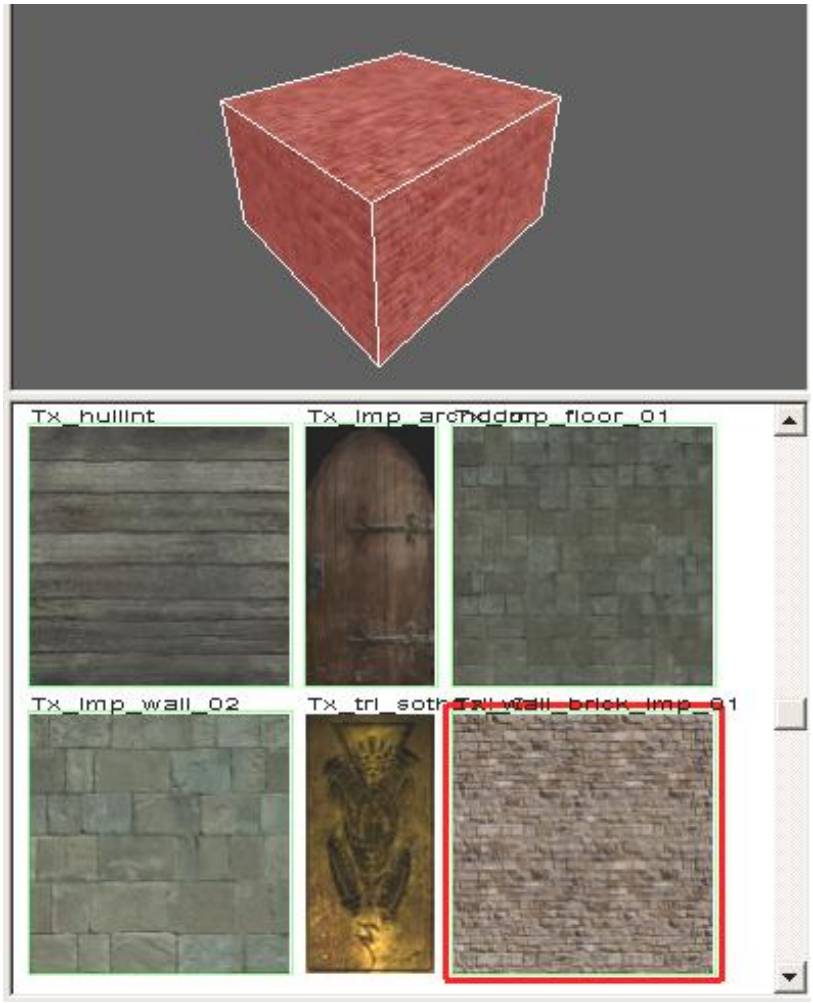
Dobrze:



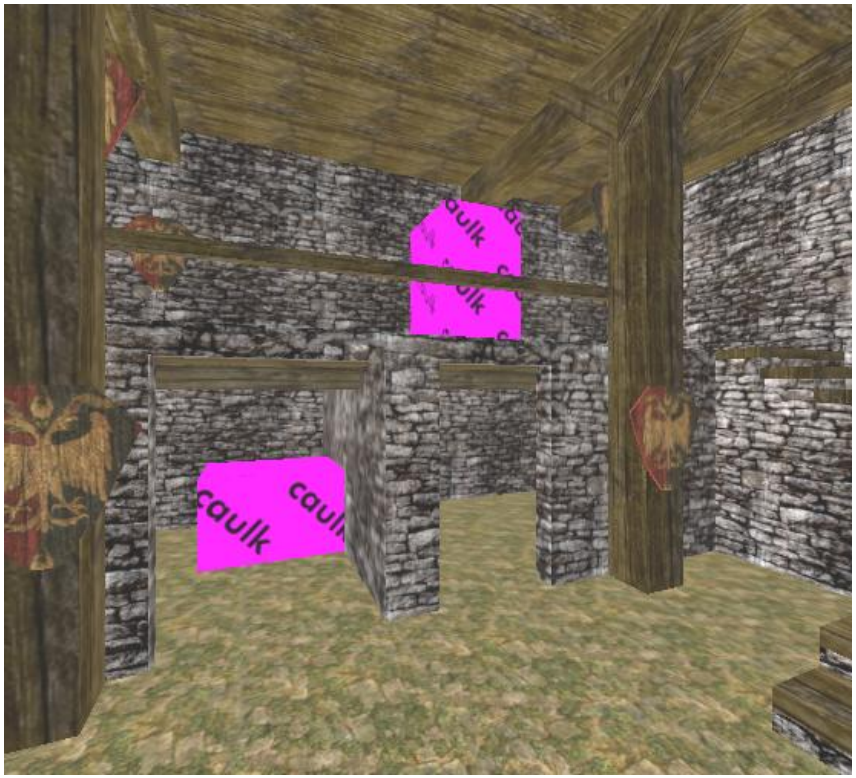
3. Bazowe bryły można deformować



4. Ściany pomieszczenia należy zateksturować



5. Niektóre obiekty można pokryć „niewidzialną” teksturą – stają się niewidzialne, ale liczone dla nich są kolizje



6. W strukturę pliku BSP można włączyć np. pliki MD2 zawierające ruchome elementy

Awatar:

Skrzynia:

Wybrane darmowe silniki graficzne

Składniki silnika

- Graf Sceny
- Pliki konfiguracyjne
- Modele i Aktorzy
- Atrybuty renderowania (światła, przezroczystość, materiały)
- Teksturowanie
- Shadery
- Sterowanie kamerą
- Obsługa dźwięków
- Obsługa skryptów
- Intefrejs 2D
- Biblioteki uzupełniające:
  - System cząsteczek
  - Detekcja kolizji
  - Zarządzanie pamięcią (garbage cllection)
  - Biblioteka matematyczna
  - Biblioteki fizyczne
  - Biblioteki obsługi sieci
  - Biblioteki Sztucznej inteligencji

XNA

- XNA (Xbox New Architecture), wydany w roku 2004, środowisko zawierające wszystkie podstawowe narzędzia służące do tworzenia oprogramowania oraz gier na komputer PC, konsolę XBOX, a także Windows Phone 7.
- Stosowana biblioteka graficzna – DirectX
- Podstawowy język programowania – C#
- Formaty plików graficznych – X, FBX
- WWW: http://msdn.microsoft.com/pl-pl/xna/default%28en-us%29.aspx

Panda3D

- Panda 3D - silnik napisany w języku C++, wywodzi się z wytwórni Disney'a
- Podstawowy język programowania – Python, ew. C++
- Format modeli 3D dla Pandy 3D ma rozszerzenie *.egg, modele w takim formacie można eksportować z poziomu programu Blender eksporterem o nazwie Chicken.
- Modele eksportowane do tekstowego formatu *.egg można później skonwertować do binarnego pliku *.bam
- Dodatkowo istnieje możliwość skorzystania z trzech silników fizycznych: wbudowanego Pandy, ODE oraz nVidia PhysX.
- Platformy systemowe: Linux, Windows, Mac OS X
- Istnieje możliwość osadzenia aplikacji na bazie Panda3D na stronach WWW
- WWW: www.panda3d.org

Ogre3D

- OGRE - Object-Oriented Graphics Rendering Engine, jest silnikiem 3D napisanym w języku C++.
- Do wyświetlania grafiki może wykorzystywać OpenGL albo DirectX.
- Ogre służy tylko do wyświetlania grafiki, więc do obsługi dźwięku, sieci i fizyki należy użyć bibliotek wyspecjalizowanych w tym celu.
- Ogre obsługuje swój format modeli 3D (*.mesh), do którego istnieją eksportery dla popularnych programówgraficznych, m. in.. Milkshape3D, 3D Studio Max, Maya, Blender, Wings3D.
- Platforma systemowa: Windows, Linux, Mac OS X
- WWW: http://www.ogre3d.org/

Crystal Space

- Crystal Space jest silnikiem graficznym napisanym w języku C++, korzystającym z OpenGL, obsługującym shadery, systemy oświetlenia, fizykę, dźwięk w przestrzeni 3D, nakładanie (mieszanie) animacji i wiele innych aspektów grafiki 3D.
- Istnieje możliwość załadowania modeli w popularnych formatach tj. *.3ds (3D Studio Max), *.cal3d (Cal3D - biblioteka do animacji szkieletowych), *.md2 (format Quake'a II).
- Za pomocą Blendera można eksportować modele do formatu Crystal Space przy użyciu pluginu blender2crystal.
- Do fizyki zostało przygotowane wsparcie dwóch silników fizycznych – Bullet oraz ODE.
- Platformy systemowe: Windows, Linux, Mac OS X, UNIX
- WWW: http://www.crystalspace3d.org

Wybrane triki grafiki 3D stosowane w grach

Motywacja

- Użytkownicy oczekują poziomu odwzorowania wirtualnego świata na poziomie zbliżonym do rzeczywistości (Final Fantasy, Ekspres polarny, Opowieść Wigilijna 2009, Avatar)
- Możliwości interaktywnej grafiki 3D (zwłaszcza na platformę PC) są wciąż ograniczone.

Gra

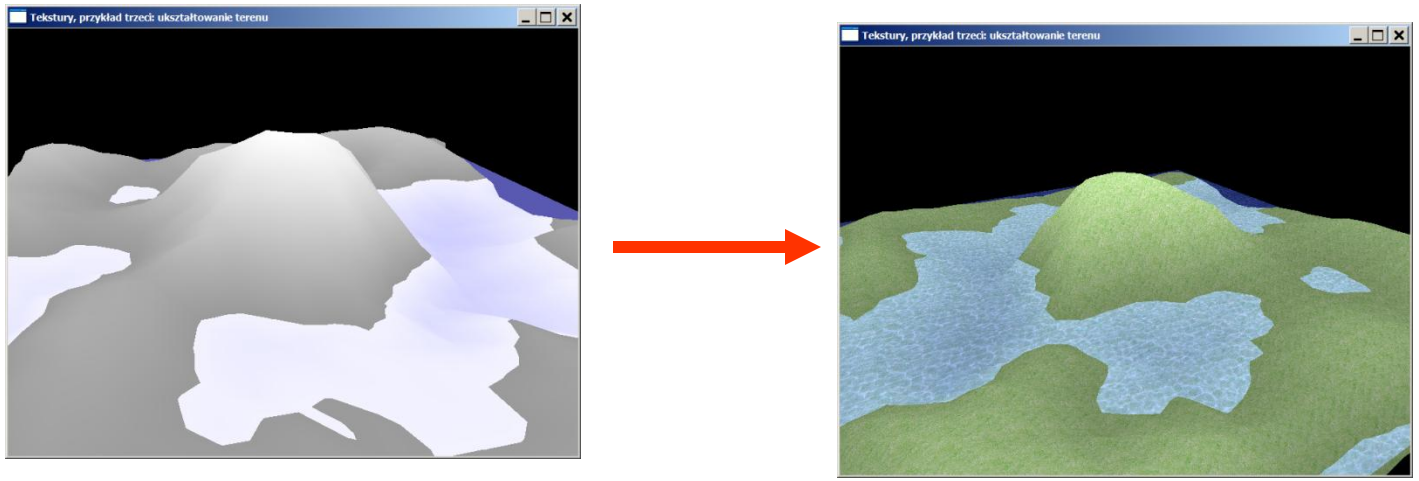
Film

Uwaga: KONSOLE!!!

- Poziom grafiki w nowych grach konsolowych przewyższa gry na PC !

Teksturowanie - dyskusja

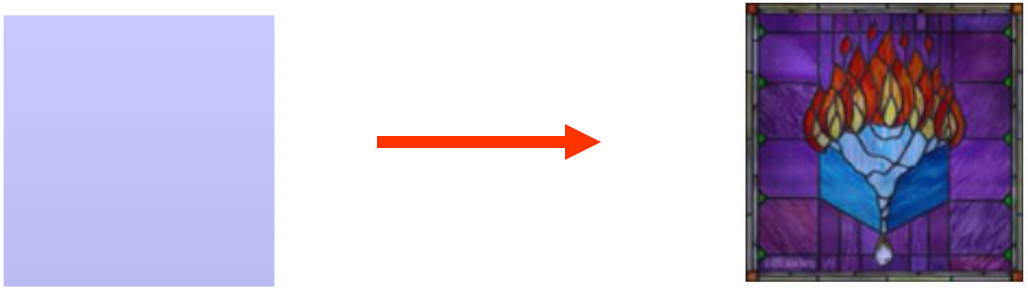
- Teksturowanie polega na pokryciu obrazem (1D/2D/3D) wielokąta
- Skomplikowane modele mogą być zastąpione prostszymi pokrytymi teksturami
- Inteligentne teksturowanie może zastąpić liczenie oświetlenia
- Jednym z głównych nurtów rozwoju kart graficznych jest powiększanie możliwości lokalnego gromadzenia tekstur i szybkiego teksturowania obiektów na scenie
- Nawet proste teksturowanie zwiększa realność sceny 3D.



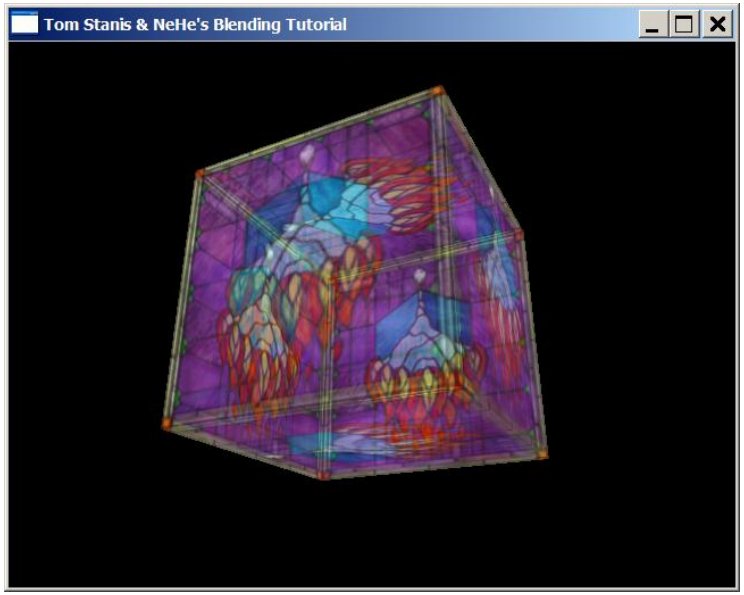
Podstawowe teksturowanie

- Załaduj plik graficzny
- Skonwertuj plik graficzny na teksturę
- Ustal filtrowanie
- Włącz teksturowanie
- Połącz współrzędne tekstury z wierzchołkami obiektu graficznego

```
glBegin(GL_QUADS);
    glTexCoord2d(1.0,1.0);glVertex3d(25,25,25);
    glTexCoord2d(0.0,1.0);glVertex3d(-25,25,25);
    glTexCoord2d(0.0,0.0);glVertex3d(-25,-25,25);
    glTexCoord2d(1.0,0.0);glVertex3d(25,-25,25);
glEnd();
```



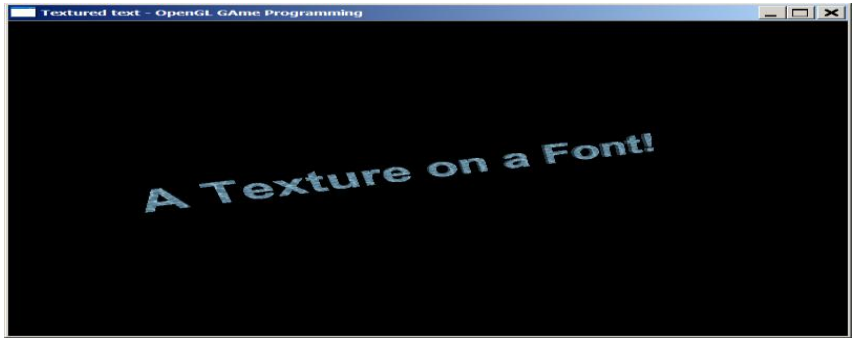
Teksturowanie z przezroczystością



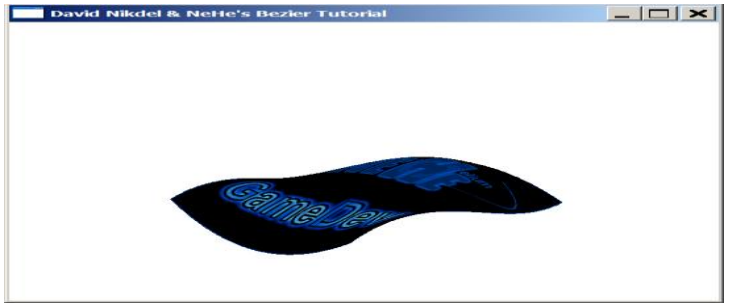

```
//…
glDepthFunc(GL_LEQUAL);
glBlendFunc(GL_SRC_ALPHA,GL_ONE);
//…
if(blend)
{ glEnable(GL_BLEND);

  glDisable(GL_DEPTH_TEST);
}
```

Możliwości teksturowania w OpenGL

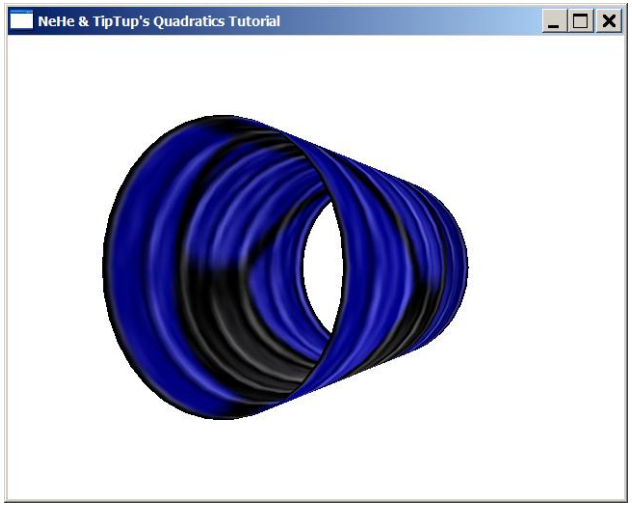
Fonts:



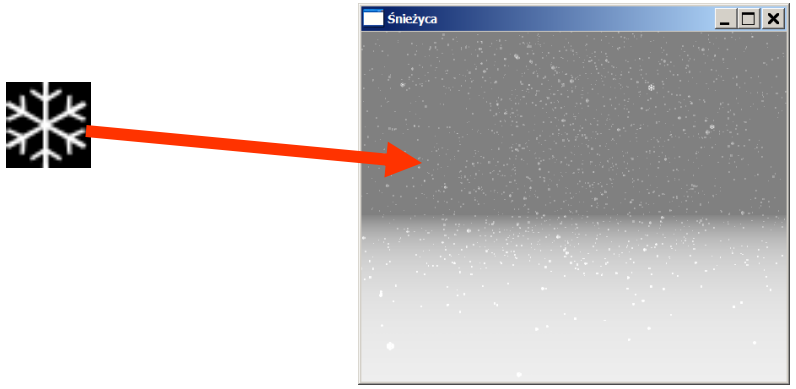
OpenGL surfaces (powierzchnie):



OpenGL quadrics (Kwadryki):



Particles (cząsteczki):

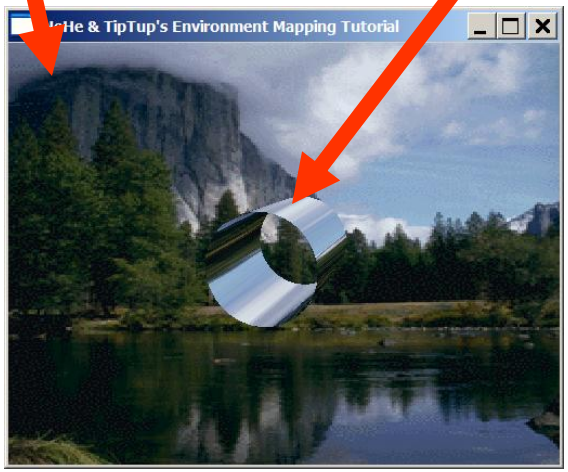


Environment Mapping

- Udawane odbicia (gry samochodowe):

Normalny obrazek:

Obrazek zniekształcony sferycznie



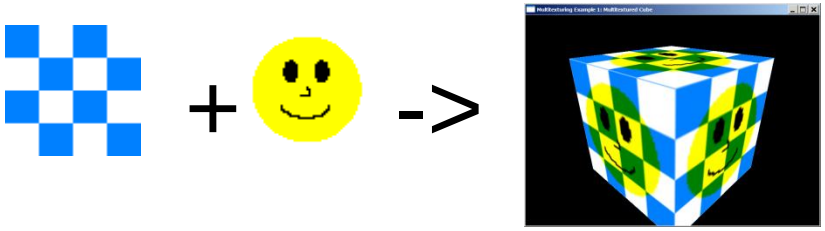

```
// THE MOST IMPORTANT COMMANDS:
//…
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,
GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,
GL_SPHERE_MAP);
//…
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
//
// gluCylinder(…);
//
glDisable(GL_TEXTURE_GEN_S);
glDisable(GL_TEXTURE_GEN_T);
```

Multiteksturowanie

- Możliwość komponowania jednej tekstury z kilku
- Dokonując tylko szybkich wymian tekstur osiąga się interesujące efekty:
  - Fałszywego oświetlenia (light maps)
  - Fałszywego odwzorowywania chropowatości (false bump mapping)

Multiteksturowanie podstawy

- Potrzebujemy 2 tekstur:

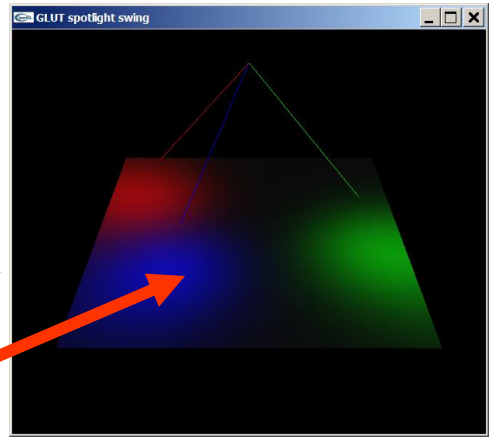


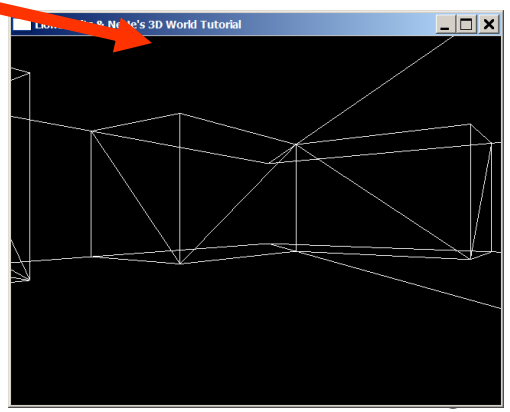
- **W OpenGL musi działać rozszerzenie GL_ARB_multitexture**:

```
glActiveTextureARB(GL_TEXTURE0_ARB); glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, smileTex->texID);
glActiveTextureARB(GL_TEXTURE1_ARB); glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, checkerTex->texID);
//…
glBegin(GL_QUADS); //…
        glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 1.0f, 0.0f);
        glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 1.0f, 0.0f);
        glVertex3f(0.5f, 0.5f, 0.5f);
//…
```

Model oświetlenia OpenGL - dyskusja

- W modelu Ponga:
  - Każdy wielokąt oświetlany jest niezależnie
  - Światło rozproszone na jednym wielokącie nie wpływa na inny
  - Cieniowanie odbywa się dla każdego wierzchołka
- Żeby uzyskać ładnie ocieniowaną powierzchnię trzeba ją podzielić na kilkadziesiąt mniejszych wielokątów.
- W grach komputerowych:
  - Dalej poszukuje się redukcji złożoności siatek
  - Nie oświetla się wszystkich elementów sceny
  - Użytkownicy oczekują realistycznych efektów
- SPRYTNIE ZASTOSOWANE MULTITEKSTUROWANIE MOŻE ZASTĄPIĆ NAKŁAD OBLICZENIOWY NA CIENIOWANIE Z ZASTOSOWANIEM OŚWIETLENIA OPENGL



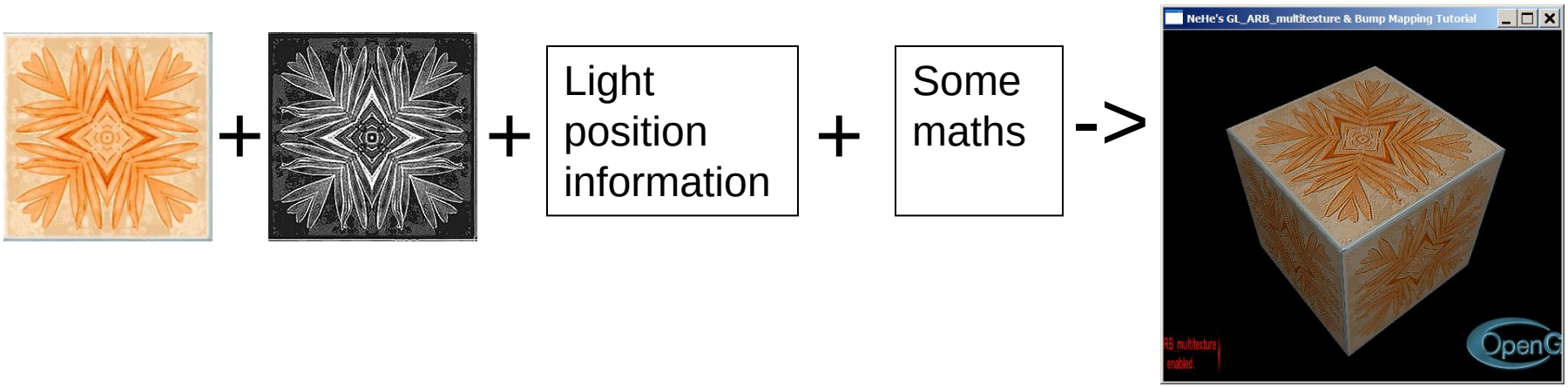


Mapy oświetlenia

- Wystarczy nałożyć na ściany kombinację tekstur odzwierciedlających fakturę powierzchni i oświetlenie, żeby uzyskać dobry efekt fałszywego oświetlenia

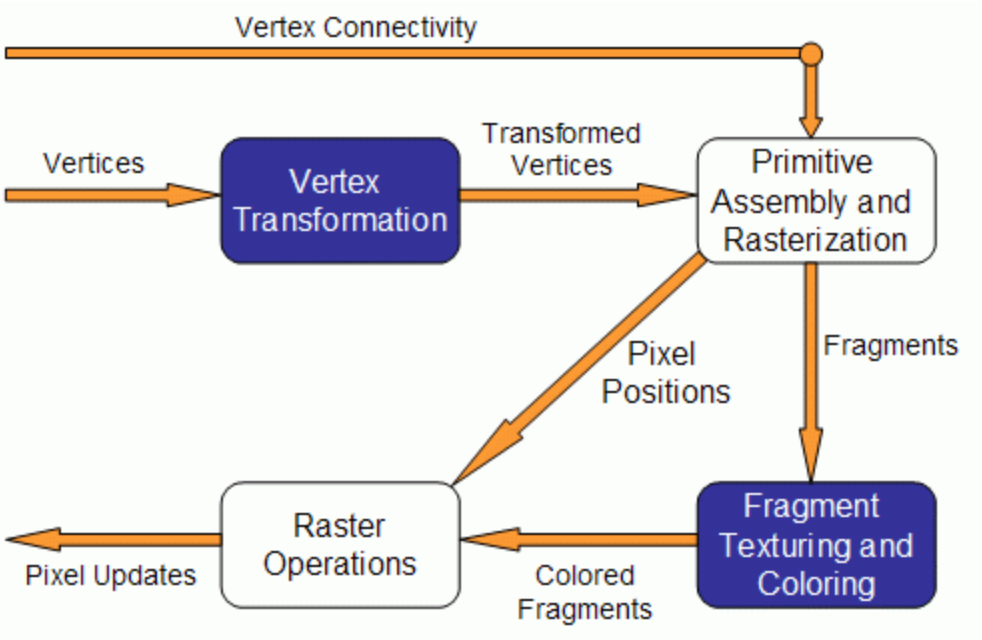
Fałszywa chropowatość

- Bada się położenie źródła światła, obserwatora oraz orientację oświetlanego obiektu
- Na podstawie wymienionych danych dokonuje się niewielkiego przemieszczenia odpowiednio przygotowanych 2 tekstur na płaszczyźnie.

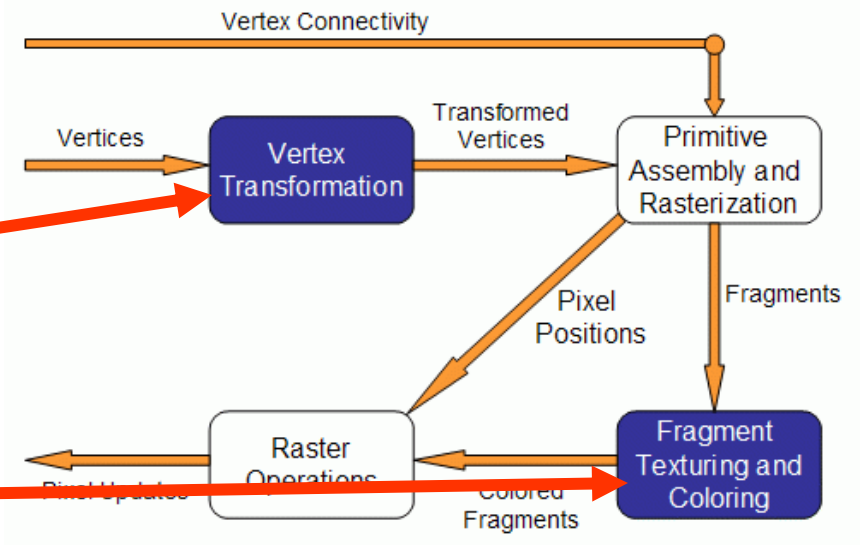


Standardowa linia potokowa OpenGL

- Transformacje wierzchołków:
  - Położenie, kolor, normalna, współrzędne tekstury
- Utworzenie prymitywów i Rasteryzacja:
  - Powiązanie wierzchołków z zastosowaniem prymitywów
  - Transformacje wycinające
  - Zdefiniowanie fragmentów – potencjalnych pikseli
- Teksturowanie fragmenntów i kolorowanie :
  - Kolory fragmentów są mieszane z kolorami tekstur
  - Obliczana jest mgła
- Operacje rastrowe:
  - Test Apha, Stencil, Głębokość
  - Ustalenie wartości koloru piksela

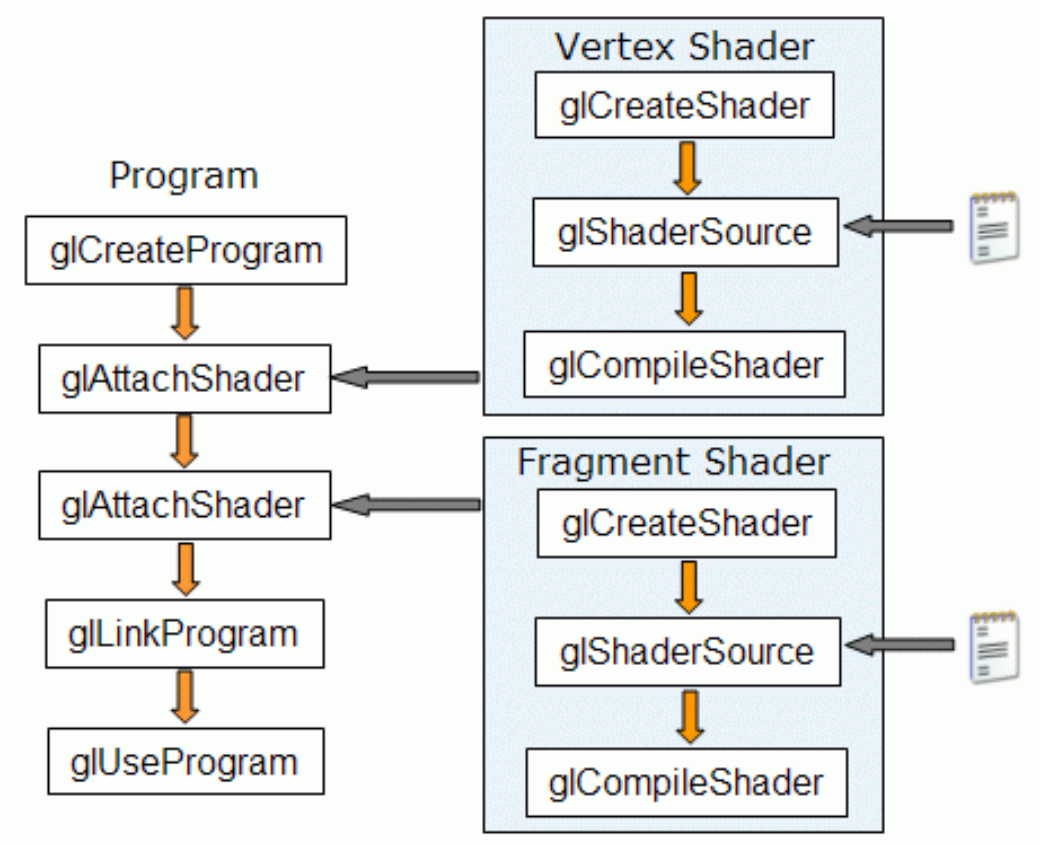


Możliwość wprowadzenia shaderów



- **Vertex shader** – możliwość modyfikowania stanu wierzchołków
- **Fragment shaders –** możliwość modyfikowania stanu fragmentów

OpenGL Shading Language (GLSL)



Technika Ray-tracing

- Droga do uzyskania lepszych wyników renderowania:
  - Obraz tworzy się przez prześledzenie promieni od źródła światła do kamery.
  - Każdy promień może wchodzić w wiele interakcji z obiektami, które stoją na jego drodze do kamery
  - Technika ray-tracing bardziej odpowiada prawom fizyki
  - Można obliczyć globalny efekt oświetlenia ale procedury są wolne i stwarzają problem w aplikacjach interaktywnych
- Istnieje ścieżka rozwoju akceleratorów graficznych, w których wbudowywane są procedury ray-tracing.