

Interakcja Człowiek-Komputer

Modelowanie postaci i otoczenia

Dr inż. Sławomir Samolej

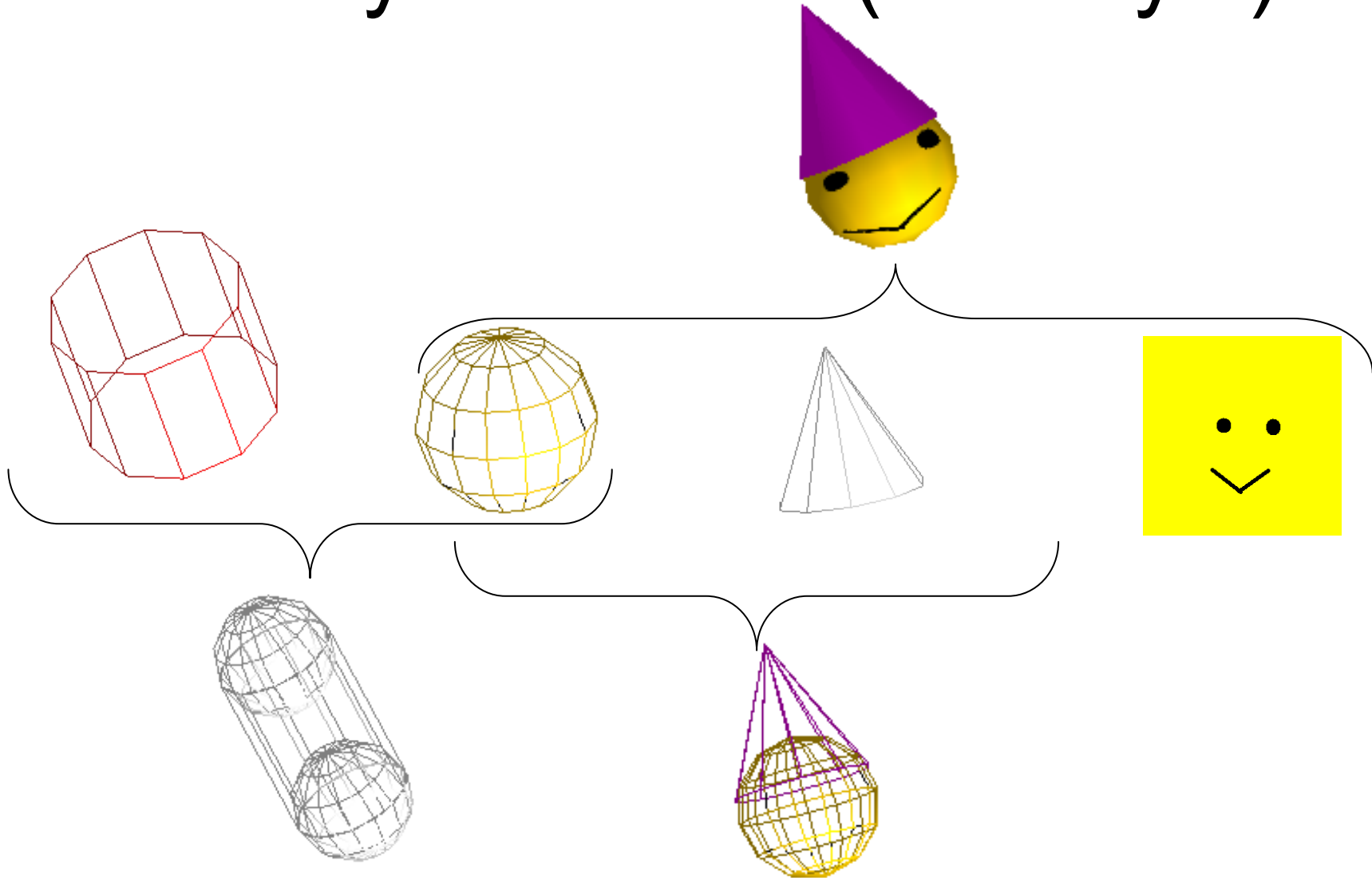
<http://ssamolej.prz-rzeszow.pl>

Literatura

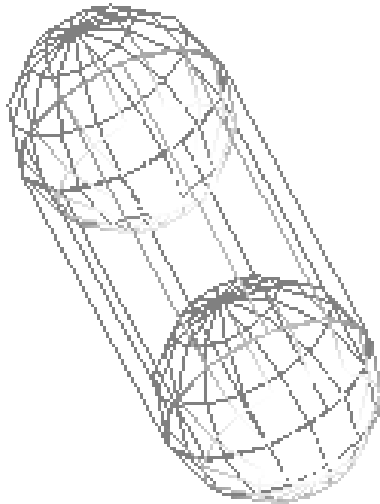
- <http://nehe.gamedev.net> – prosty silnik 3D
- <http://www.ogre3d.org/> - zaawansowany silnik 3D
- <http://www.panda3d.org/> - silnik 3D
- <http://www.blender.org/> - program do tworzenia obiektów 3D
- Kevin Hawkins and Dave Aistle, *OpenGL Game Programming*
- K. Kuklo, J. Kolmaga, *Blender Kompendium*, 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.

Modelowanie prostej postaci z zastosowaniem OpenGL

Elementy składowe (kwadryki)

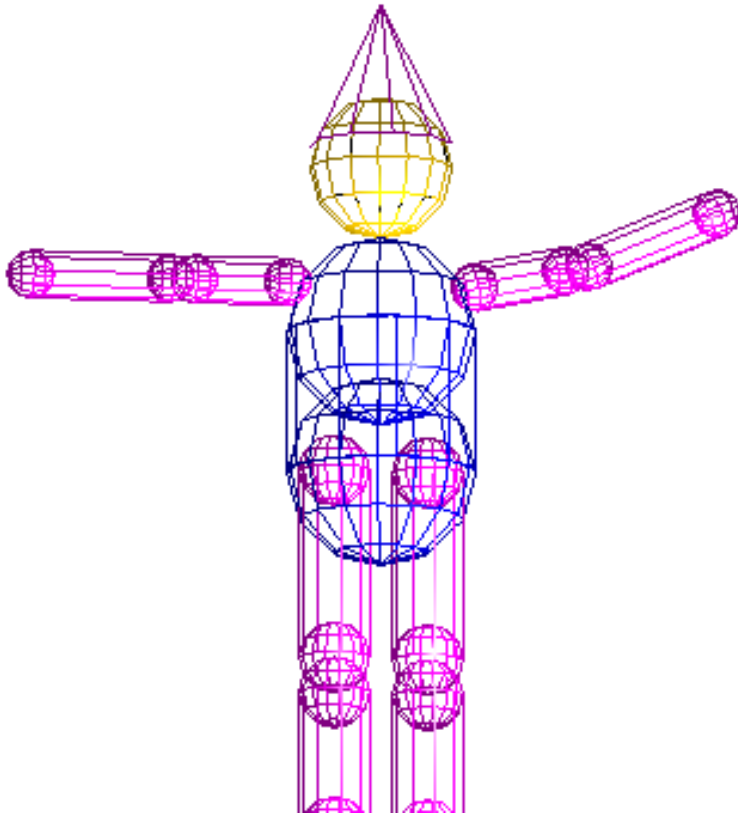


Podstawowy „moduł”



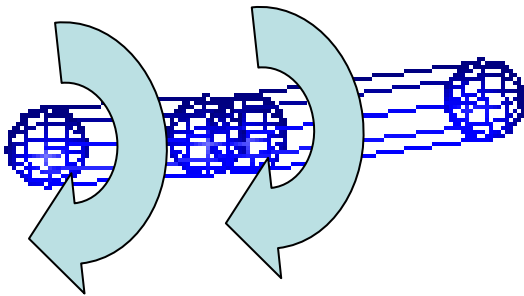
```
void Modul(double dlugosc,
           double srednica
           )
{
    static GLUQuadricObj *w1;
    w1=gluNewQuadric();
    glPushMatrix();
    gluCylinder(w1,
                srednica,
                srednica,
                dlugosc,
                10,
                1
                );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

Siatka



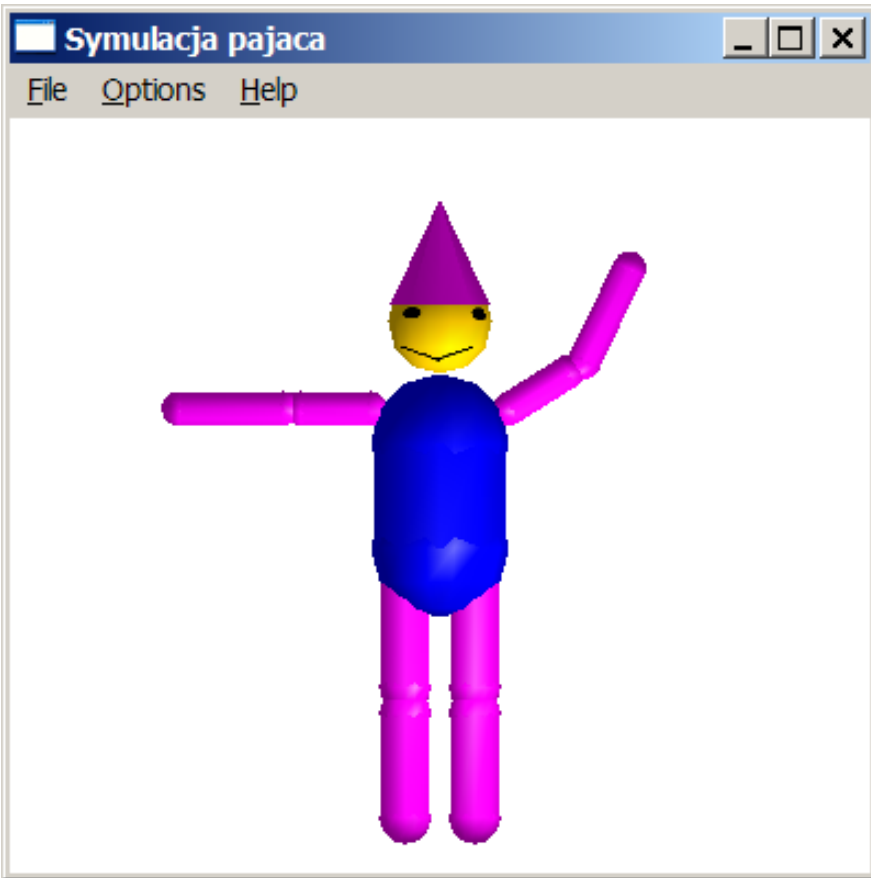
```
void Pajac(double glowa_y,  
           double glowa_z,  
           double p_ramie_z,  
           double p_ramie_x,  
           double p_przedramie_z,  
           double l_ramie_z,  
           double l_ramie_x,  
           double l_przedramie_z,  
           double sklon,  
           double 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);

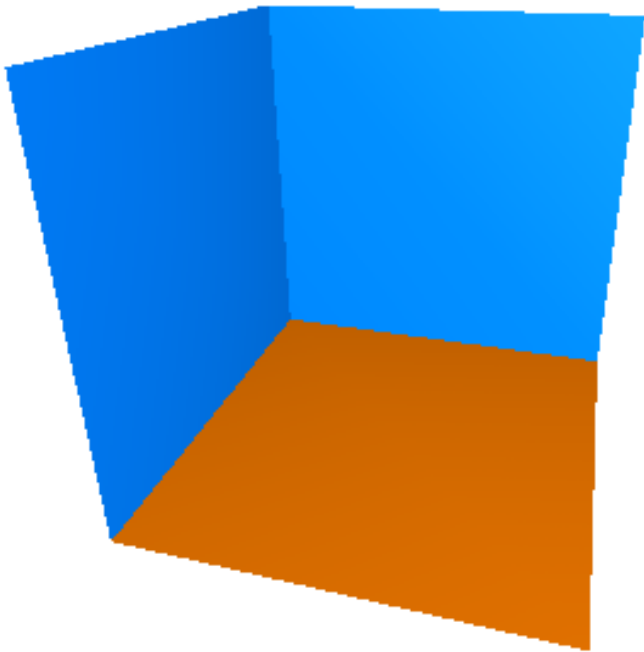
    break;
```

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
 - ...

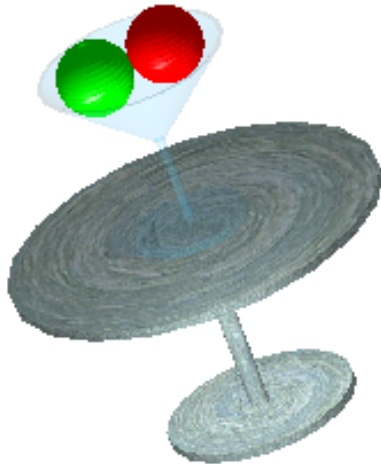
Modelowanie otoczenia z zastosowaniem OpenGL

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

Prosty prostokątny świat

- Struktura zapamiętania wierzchołków:

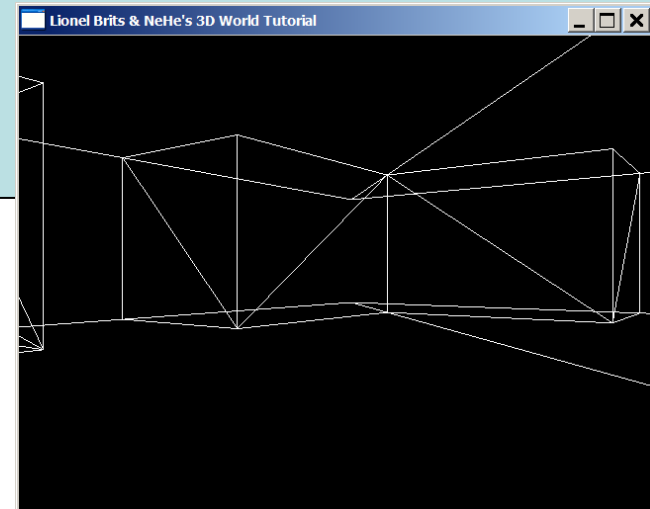
```
typedef struct tagVERTEX
{float x, y, z;           // 3D Coordinates
 float u, v;             // Texture Coordinates
} VERTEX;                // Call It VERTEX
```

- Skrypt :

```
NUMPOLYIES 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



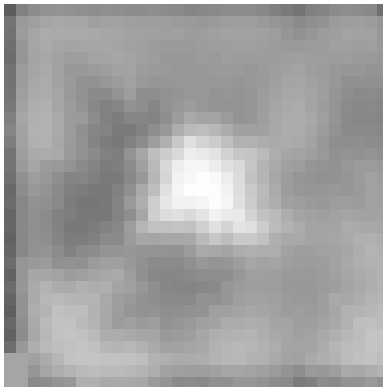
```
// Key management:
if (keys[VK_RIGHT])    // Right Arrow Being Pressed?
{ yrot -= 1.5f; }      // Rotate To The Left
if (keys[VK_LEFT])    // Left Arrow Being Pressed?
{ yrot += 1.5f; }      // Rotate To The Right
if (keys[VK_UP])       // Up Arrow Being Pressed?
{ // Move On The X-Plane Based On Player Direction
    xpos -= (float)sin(heading*piover180) * 0.05f;
// Move On The Z-Plane Based On Player Direction
    zpos -= (float)cos(heading*piover180) * 0.05f; //... }

// Rendering the scene:
glRotatef(lookupdown,1.0f,0,0); // Rotate To Look Up And Down
glRotatef(sceneroty,0,1.0f,0);  // Direction Player Is Facing
// Translate The Scene Based On Player Position
glTranslatef(xtrans, ytrans, ztrans);
```

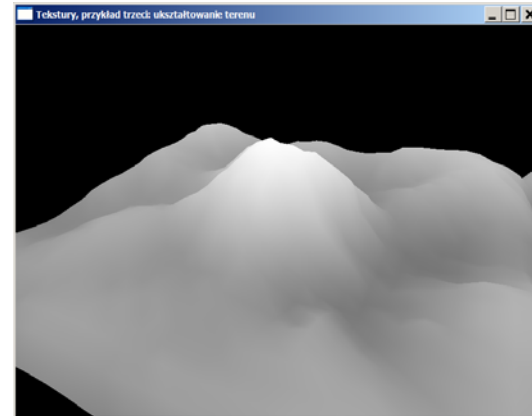
Mapy wysokości - idea

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

Bitmapa:

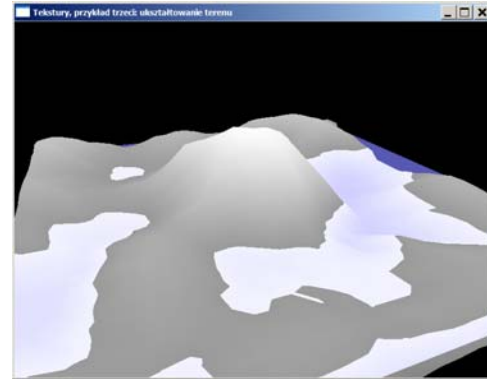
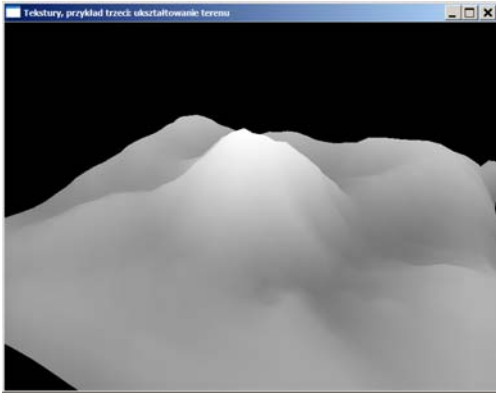


Teren:

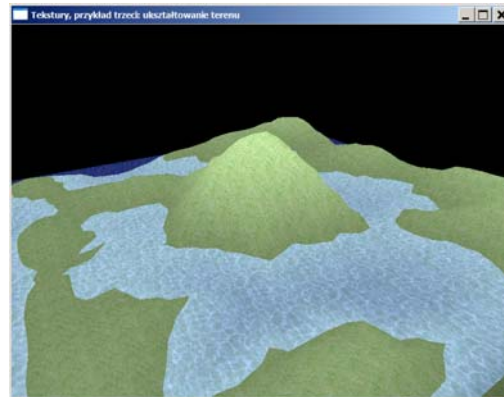


Mapy wysokości – podstawowe triki

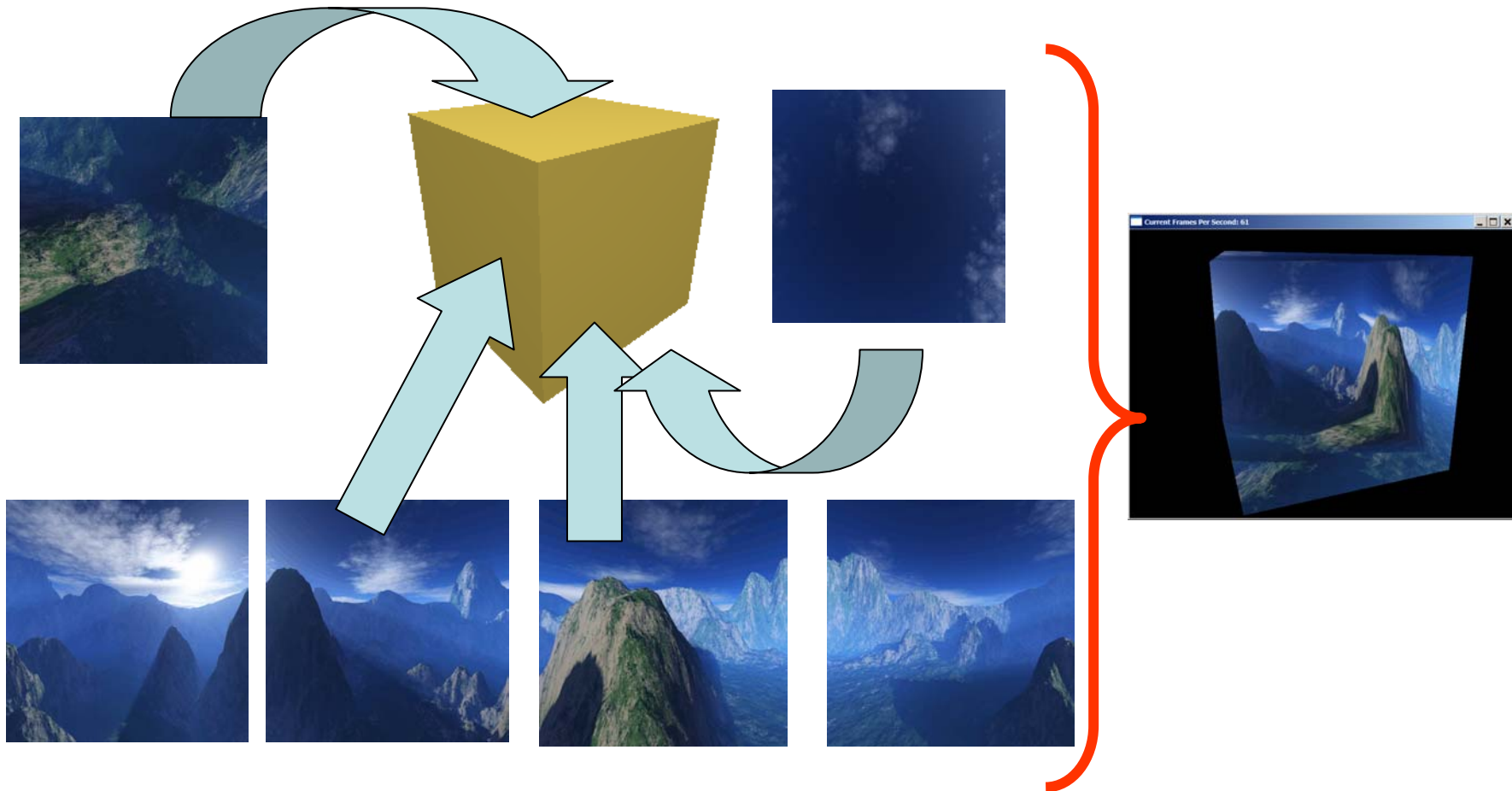
- Falująca woda:



- Tekstutowanie:

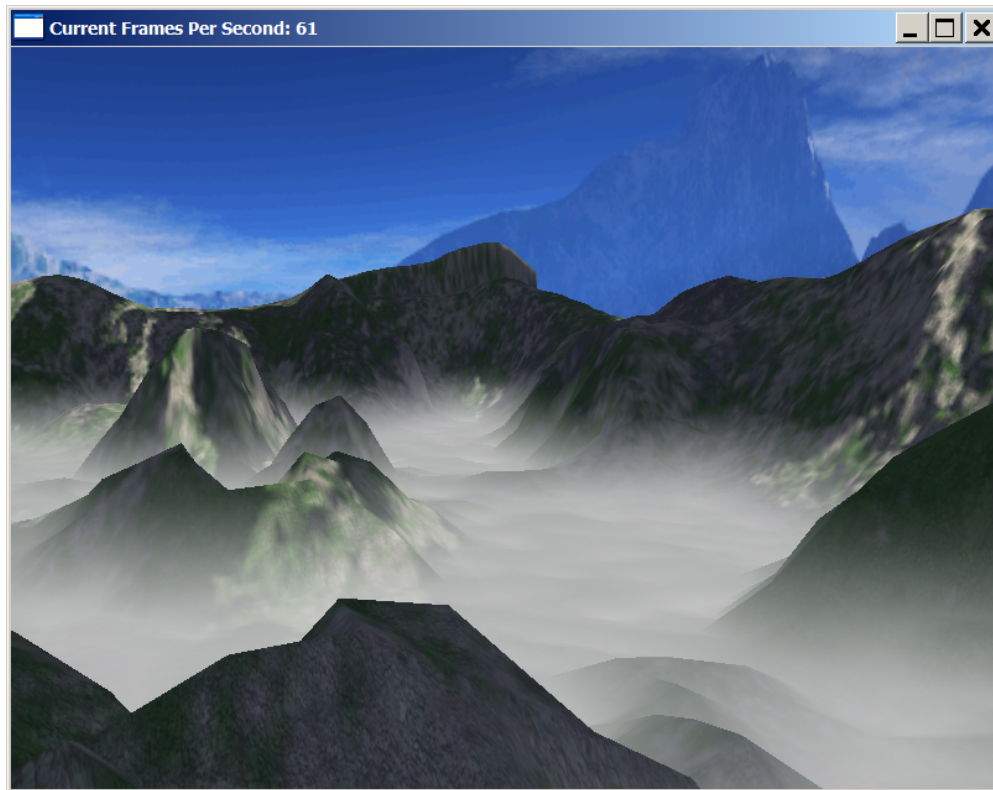


SKY BOX



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.

Inne narzędzia do tworzenia interaktywnej grafiki 3D

Wojny bibliotek interaktywnej grafiki 3D

OpenGL vs. Direct3D

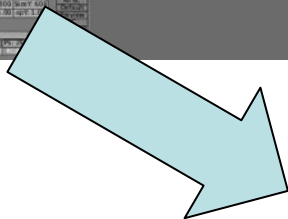
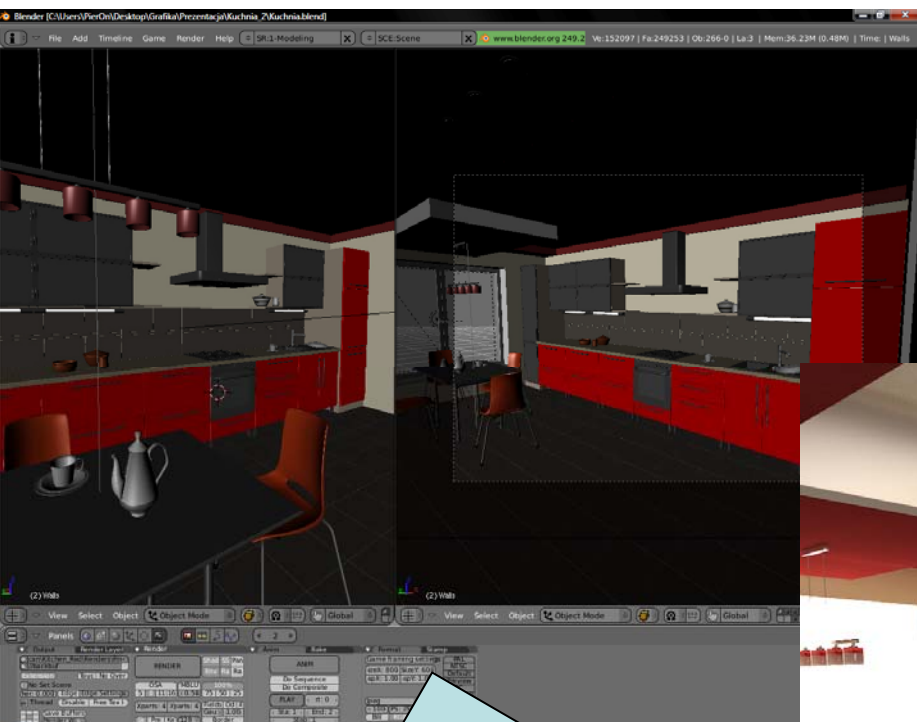
- Coraz trudniej je odróżnić.
- Nigdy próba konwersji z jednej do drugiej nie da „przyspieszenia” aplikacji

„Filmowa” i „Zdjęciowa” Grafika 3D

- 3D Studio Max
- Lightwave 3D
- Maya
- Blender

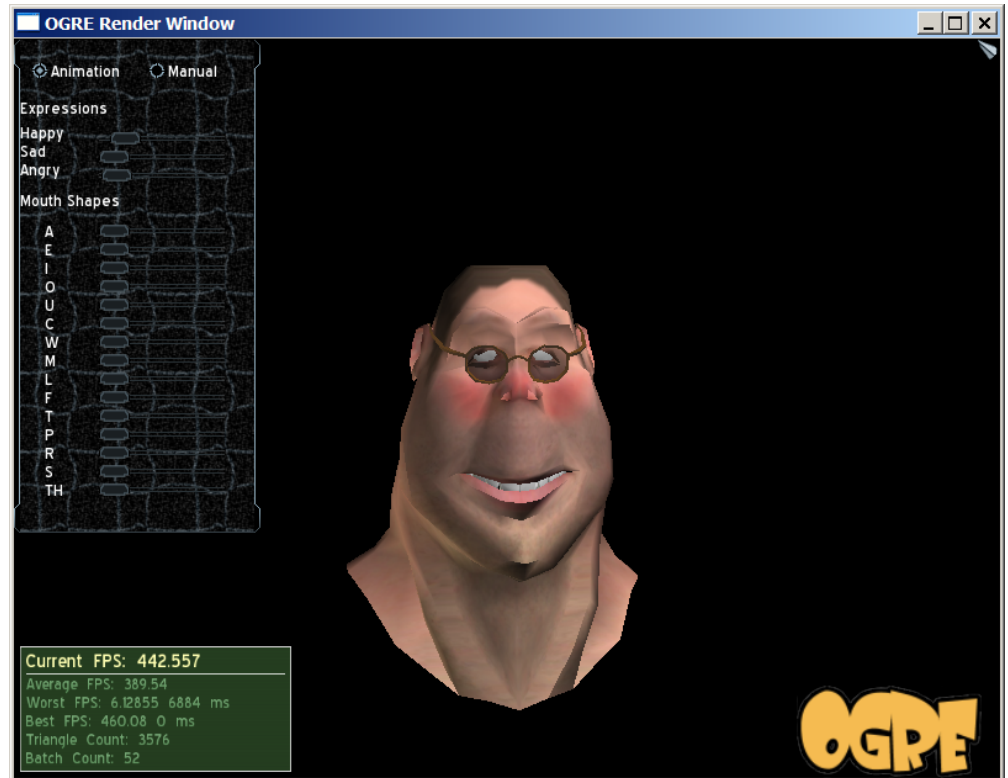


Przykład modelu przed i po renderingu



Silniki graficzne

- Panda3D
- XNA
- Dark GDK
- Ogre3D
- Crystal Space



Biblioteki graficzne vs. Silniki graficzne

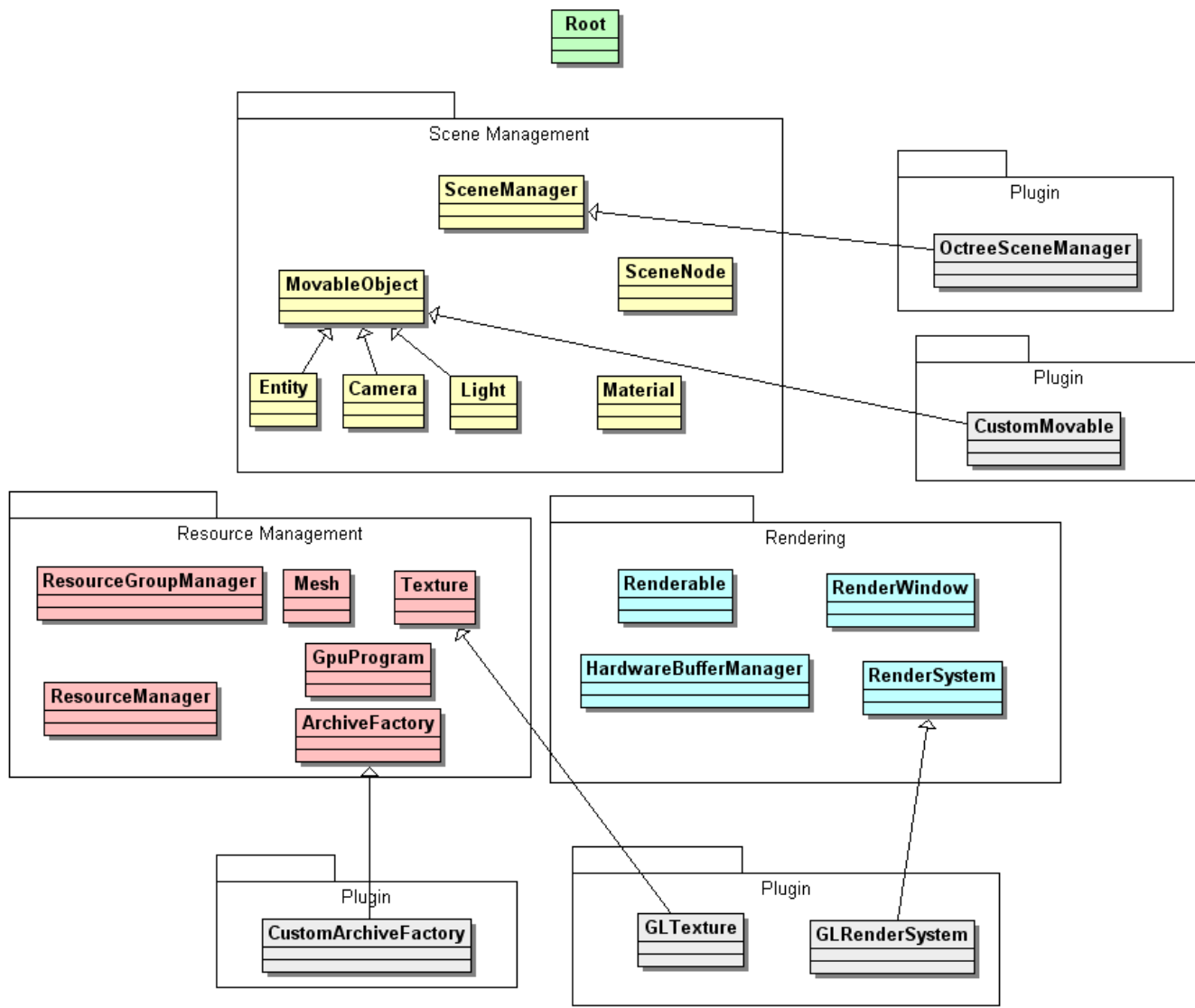
- Biblioteki „myślą” na poziomie wierzchołków, prymitywów, transformacji przestrzennych, teksturowania i shaderów, położenia kamery
- Silniki „myślą” na poziomie „postaci”, „otoczenia”, położenia kamery, shaderów.

„Inżynierskie” podejście do tworzenia interaktywnej grafiki 3D

- Zastosować „filmowe” narzędzia do utworzenia elementów sceny (uwzględniając stopień złożoności obiektów !)
- Zastosować silnik graficzny do „załadowania” elementów graficznych i zarządzania nimi na scenie



Struktura przykładowego silnika graficznego



Struktura przykładowego silnika graficznego (2)

- „Matematyka graficzna”
- Podsystem zarządzania sceną
- Podsystem wyświetlania sceny
- Podsystem „ładowania” plików
- Podsystem „sterowania”
- Rozszerzenia:
 - Komunikacja sieciowa
 - Odtwarzanie dźwięków
 - Modelowanie fizyki

Wybrane formaty plików graficznych do reprezentacji postaci

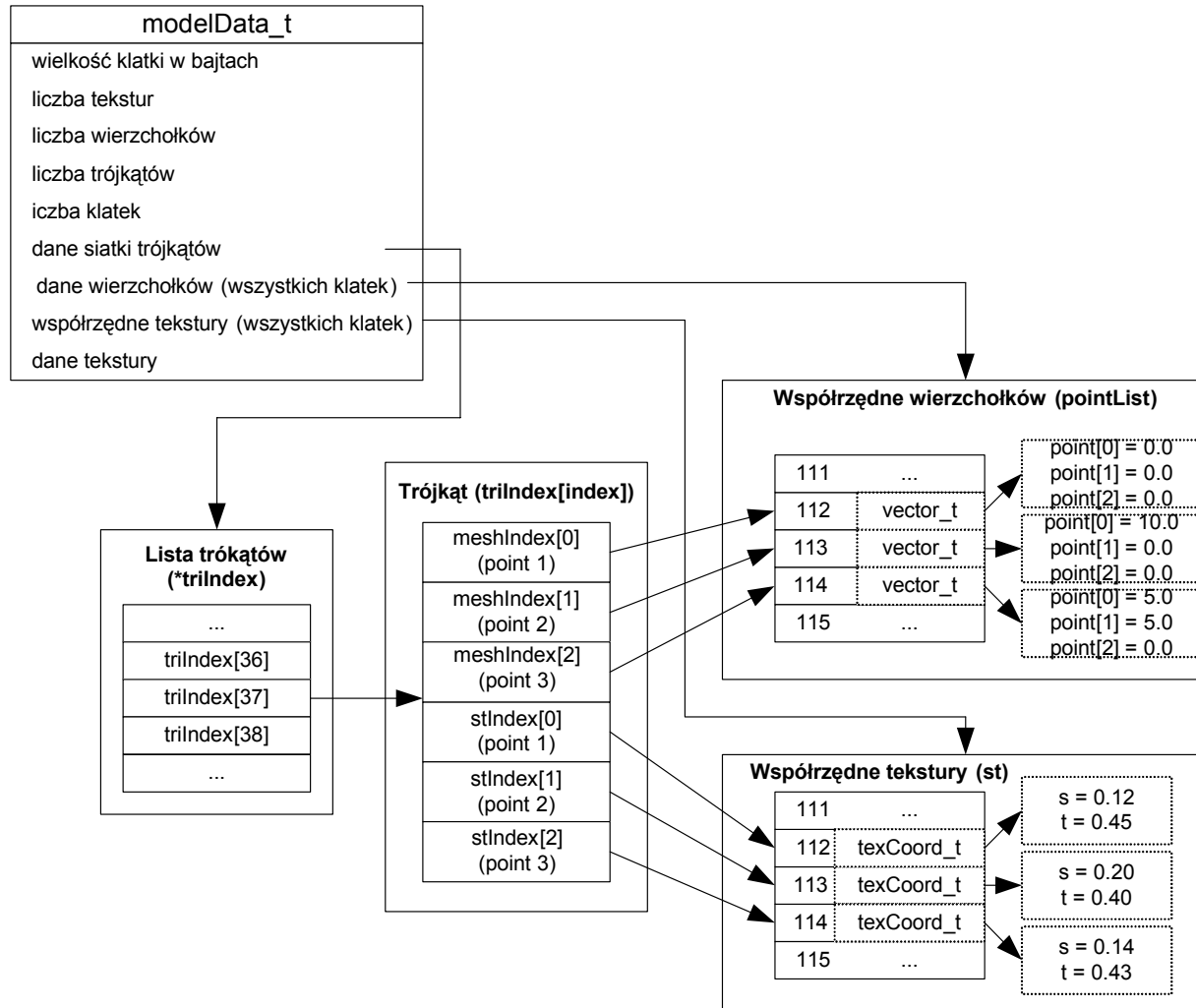
Wybrane formaty plików graficznych do reprezentacji postaci

- MD2 - nagłówek

```
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;
```

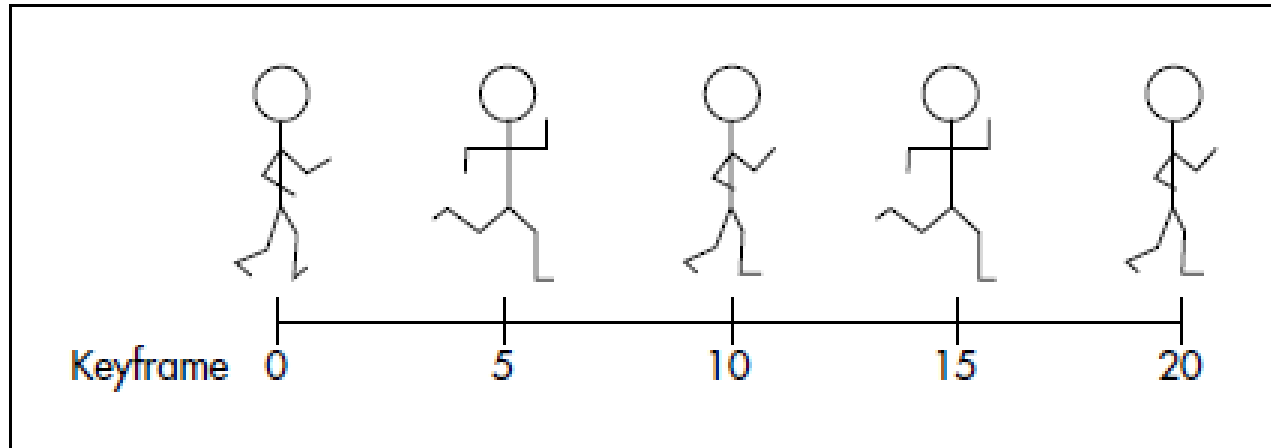
Wybrane formaty plików graficznych do reprezentacji postaci

- MD2 – interpretacja danych



Wybrane formaty plików graficznych do reprezentacji postaci

- Klatki kluczowe



Wybrane formaty plików graficznych do reprezentacji postaci

- MD3

- 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



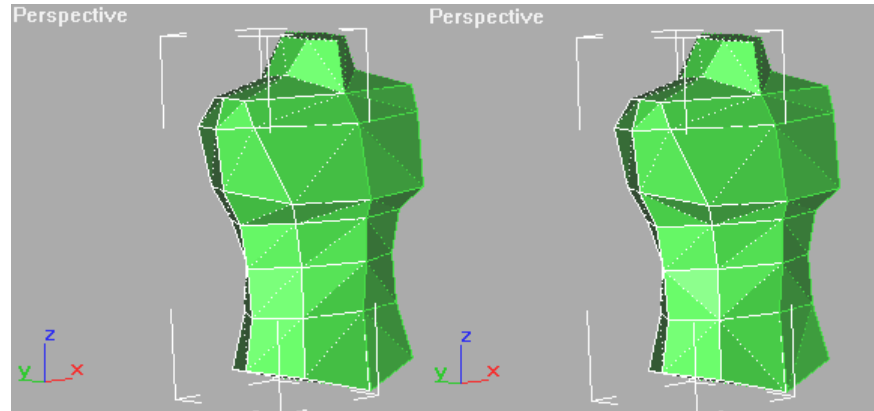
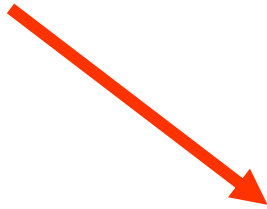
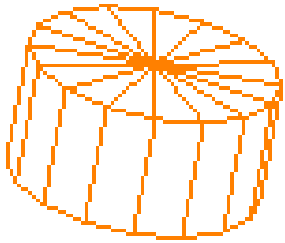
Wybrane formaty plików graficznych do reprezentacji postaci

- Format Mesh Ogre3D
 - Własny format Ogre w postaci pliku XML
 - Zawiera informacje o wierzchołkach, teksturach i kościach powiązanych z grupą wierzchołków

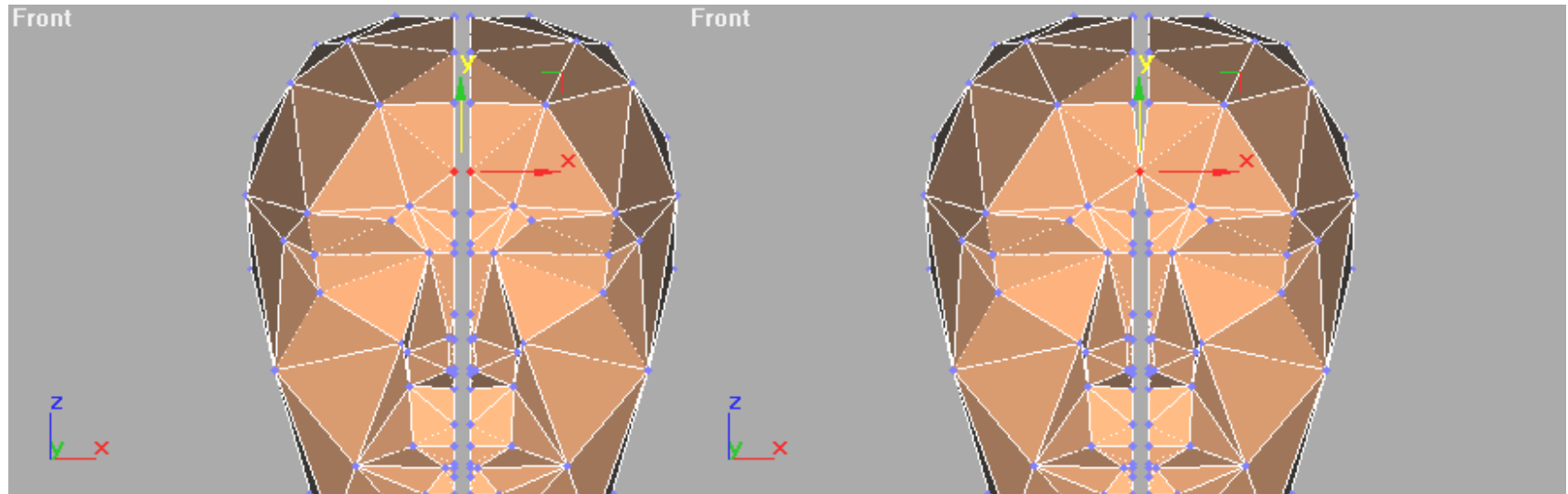


Zasady tworzenia zaawansowanej postaci

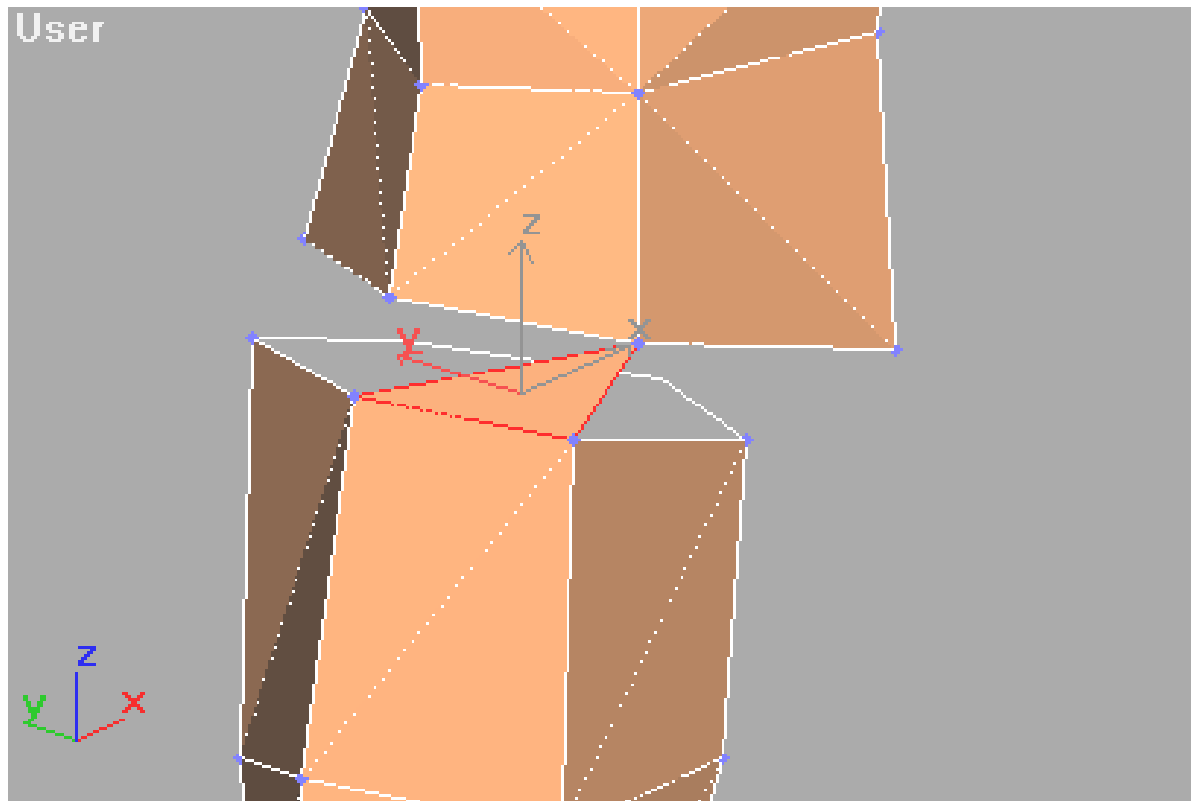
1. Modyfikowanie bazowej siatki



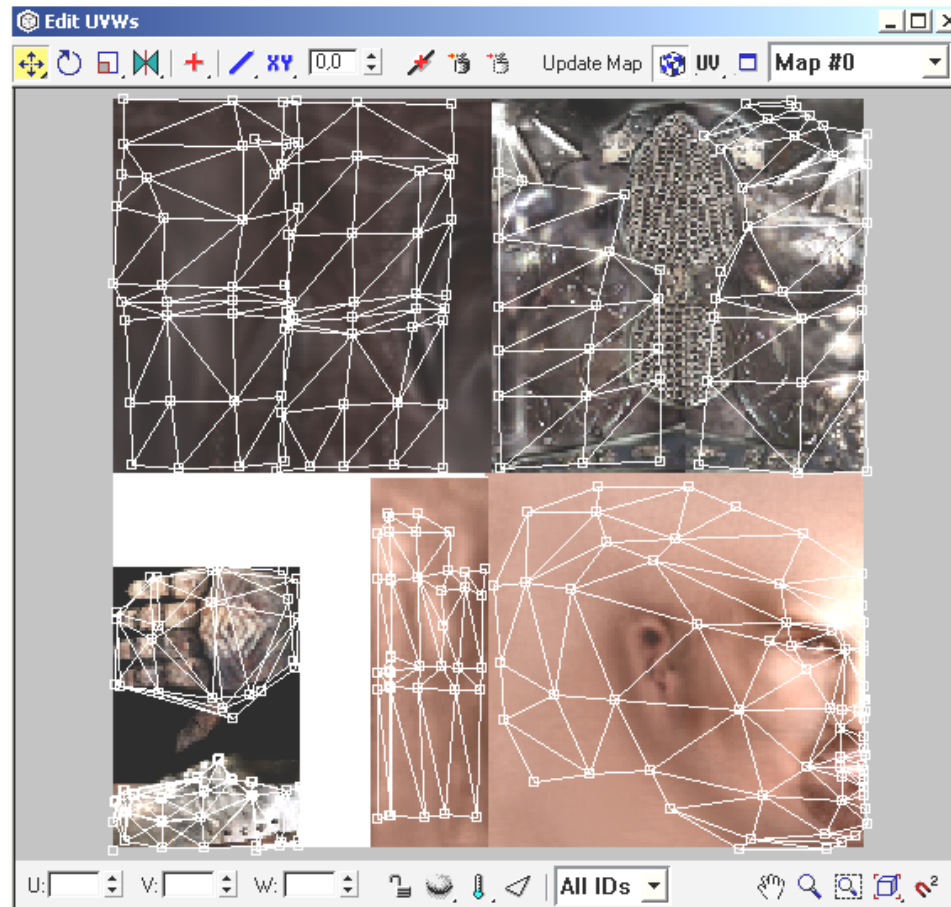
2. Buduje się połowę postaci – drugą otrzymuje się prze „odbicie” i „sklejenie”



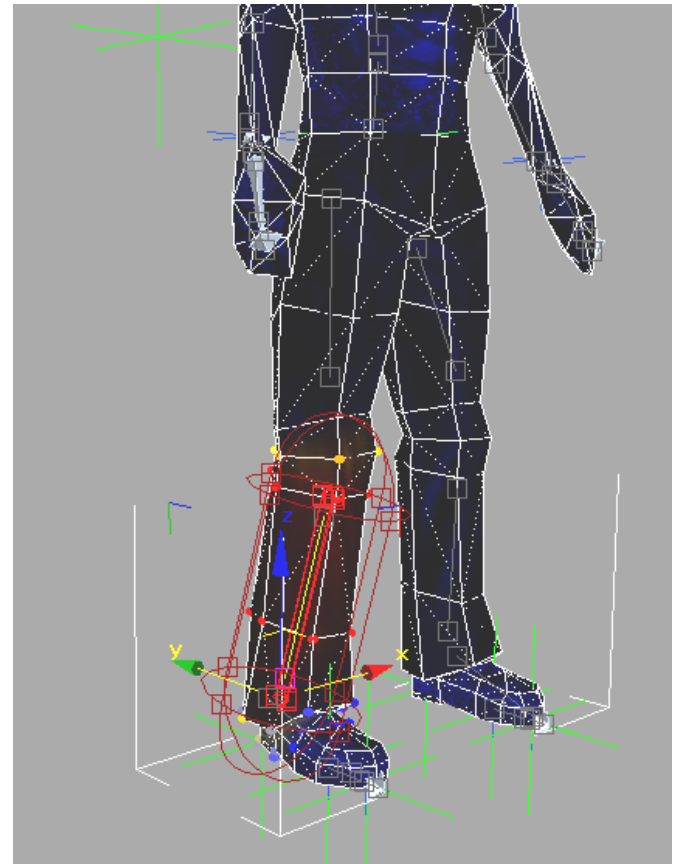
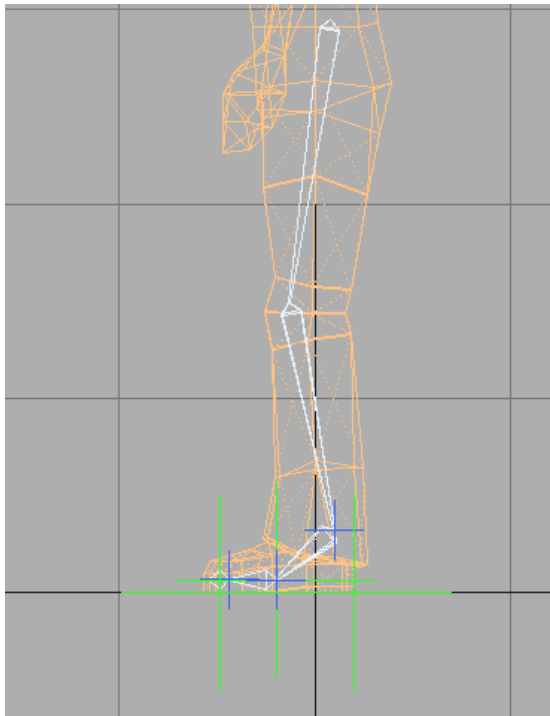
3. Jeśli postać składa się z kilku oddzielnych siatek, trzeba je „skleić”



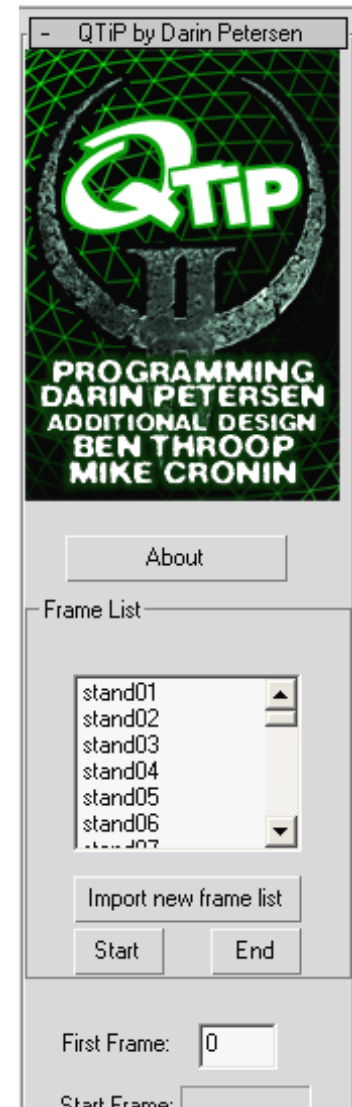
4. Z węzłami siatki trzeba połączyć odpowiednie współrzędne tekstur



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

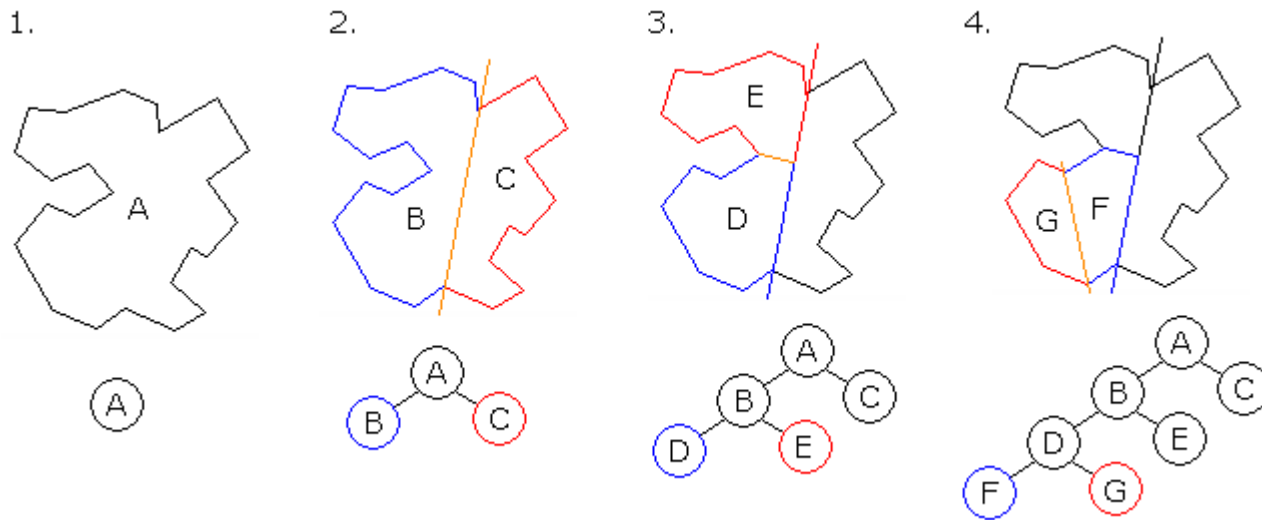


Metody budowania zaawansowanego „otoczenia”

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 określających 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

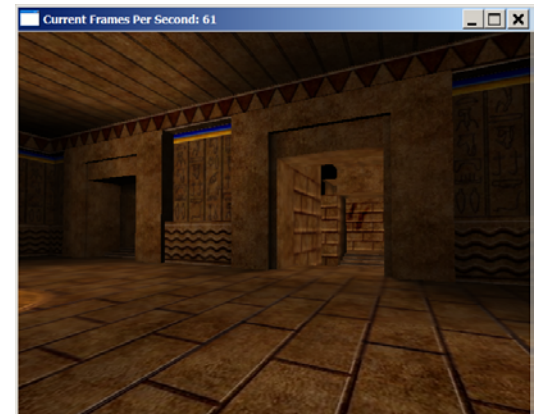
- Zaproponowany przez ID Software (DOOM, QUAKE),
- Stosuje algorytm tworzenia drzew BSP do podziału przestrzeni na wypukłe wielościany
- Przechowuje informację o geometrii otoczenia oraz innych jego składnikach (dodatkowych obiektach, mapach oświetlenia, teksturach, shaderach itp.)
- Nagłówek pliku składa 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

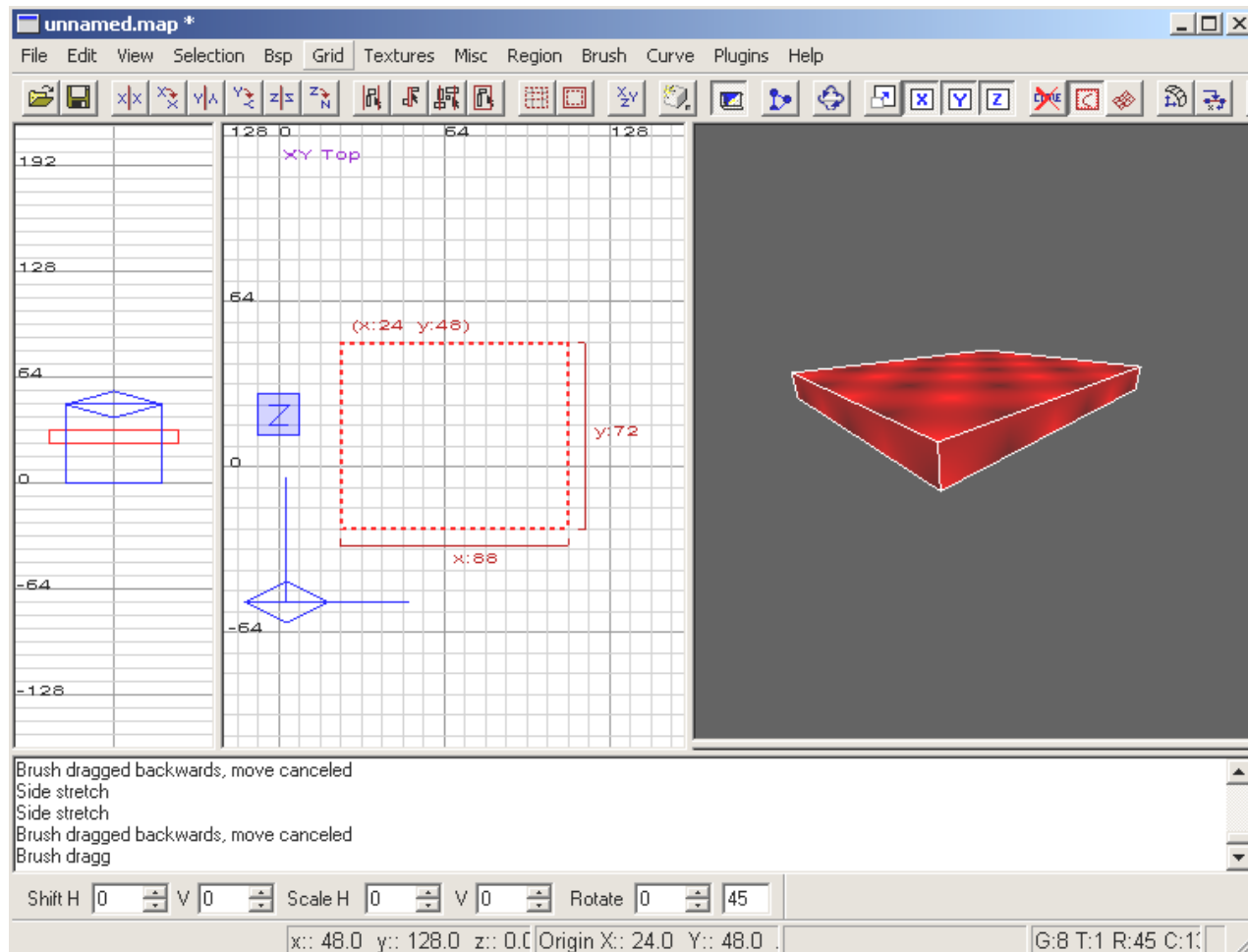
Quake BSP – praktyczne uwagi

- Pliki Quake BSP można tworzyć z zastosowaniem darmowego oprogramowania Q3Radiant
- Zastosowanie plików Quake BSP 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



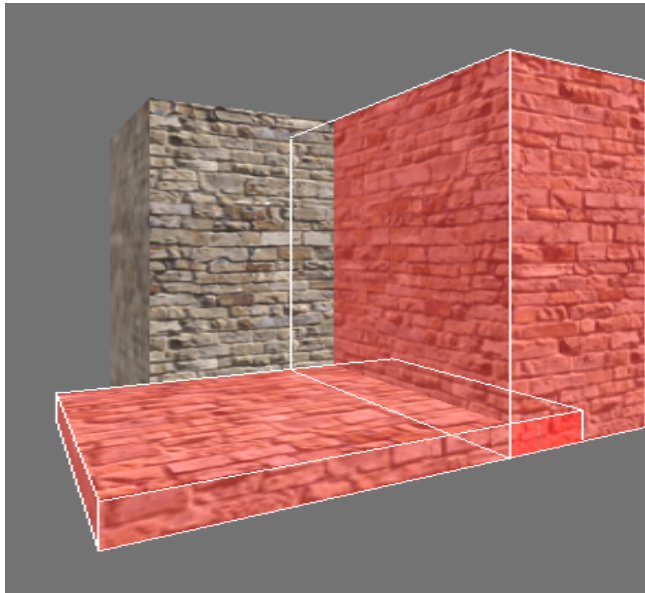
Zasady budowania modelu otoczenia w Q3Radiant

1. Tworzenie modelu zaczyna się od prostopadłościanu

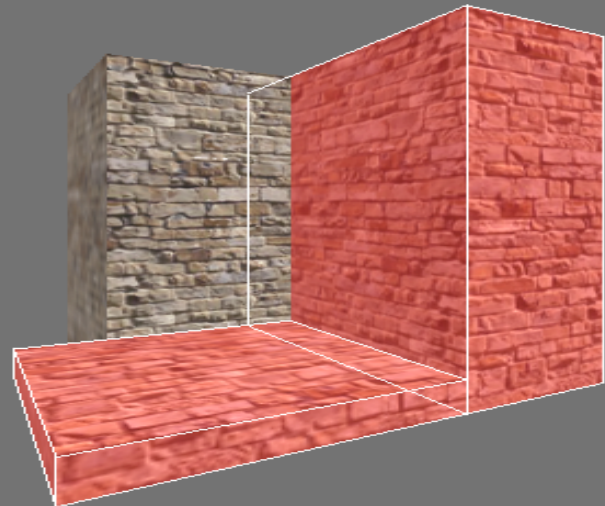


2. Bryły nie mogą na siebie zachodzić, prostopadłościany można skalować i obracać

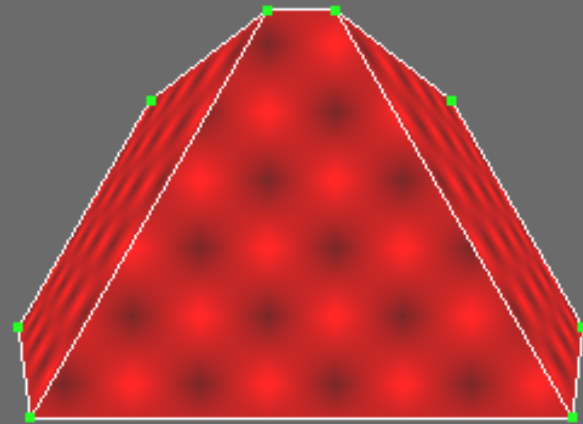
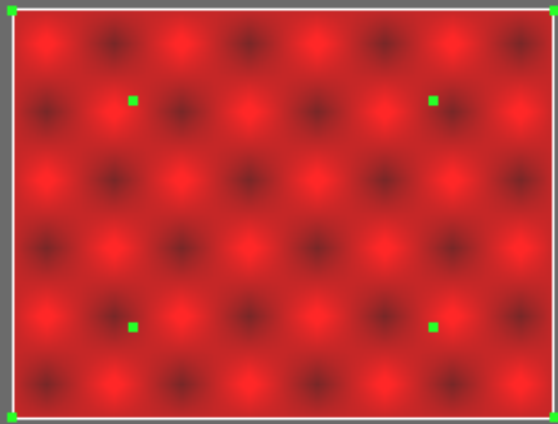
Źle:



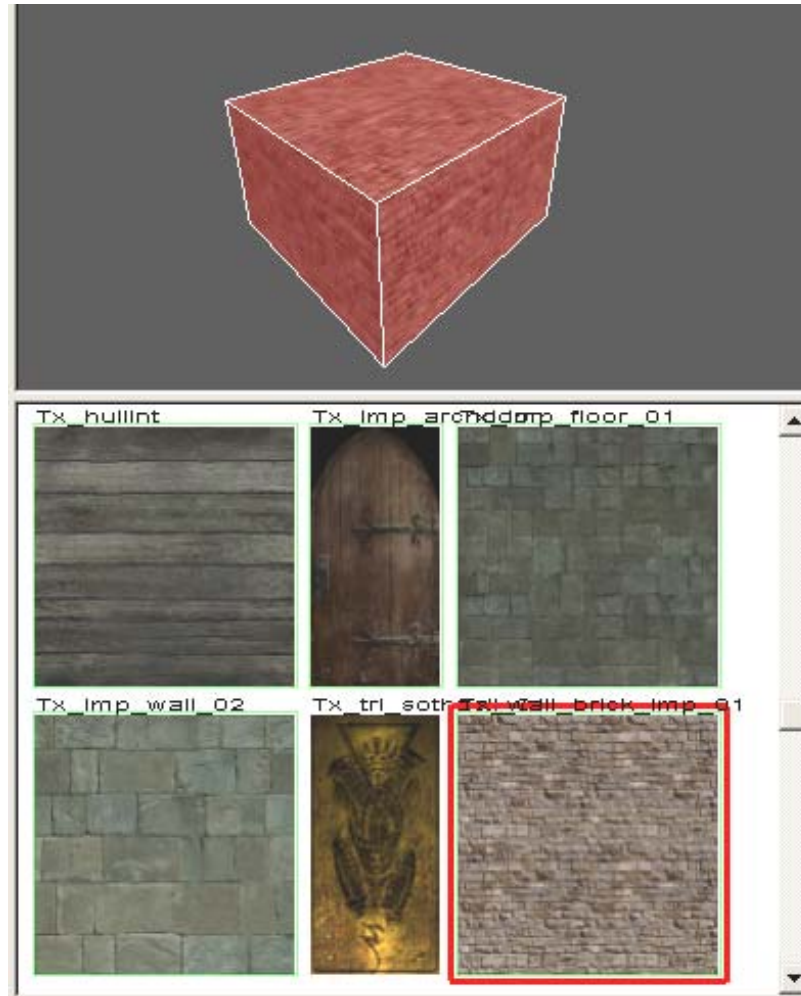
Dobrze:



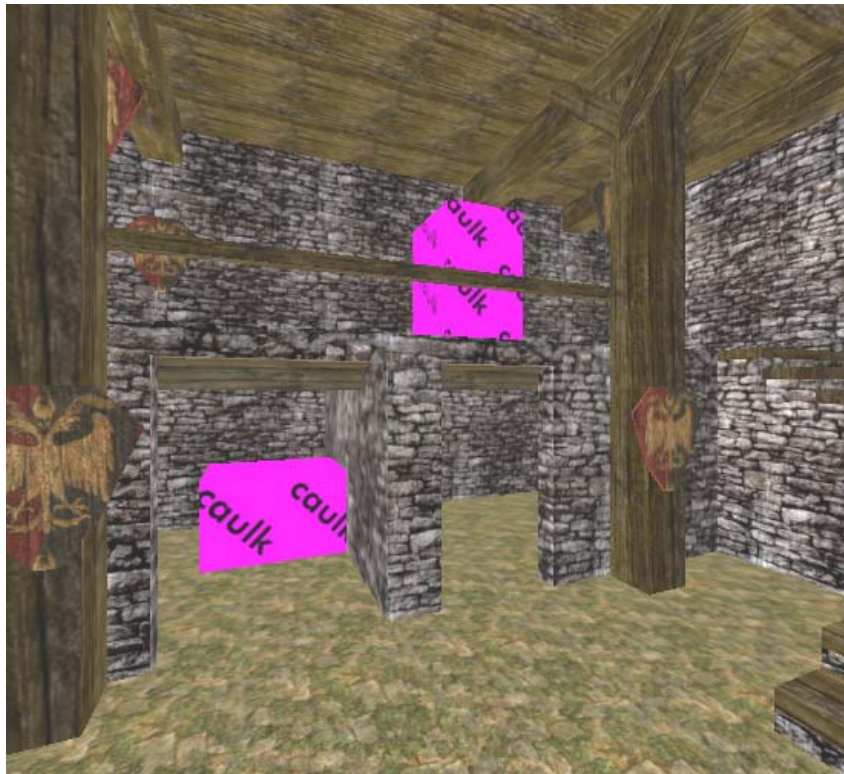
3. Bazowe bryły można deformować



4. Ściany pomieszczenia należy zateksturuować



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:



Przykład – prosta sieciowa gra przygodowa

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



Przykład – wirtualna KIA

- Poziom – Quake BSP
- Silnik DarkGDK



Przykład – wirtualna postać do interakcji z dziećmi



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)
- Możliwości interaktywnej grafiki 3D (zwłaszcza na platformę PC) są wciąż ograniczone.



Gra



Film

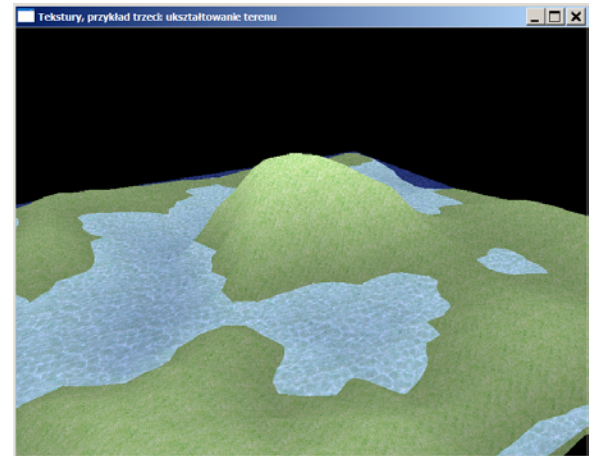
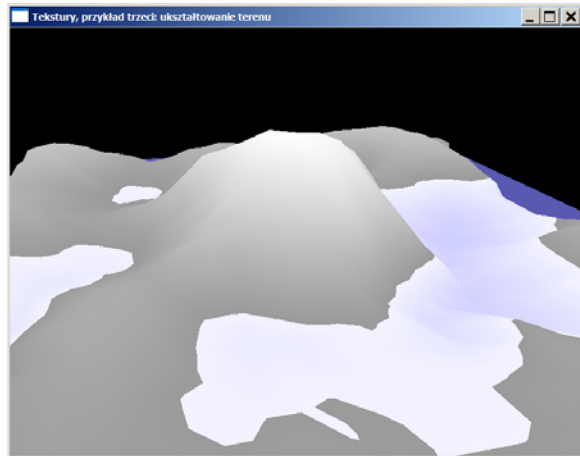
Uwaga: KONSOLE!!!

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



Teksturowanie - dyskusja

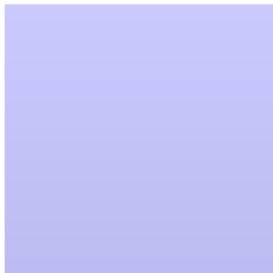
- Teksturowanie polega na pokryciu obrazem (1D/2D/3D) wielokąta
- Skomplikowane modele mogą być zastąpione prostszymi pokrytymi teksturami
- Inteligentne teksturowanie może zastąpić liczenie oświetlenia
- Jednym z głównych nurtów rozwoju kart graficznych jest powiększanie możliwości lokalnego gromadzenia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



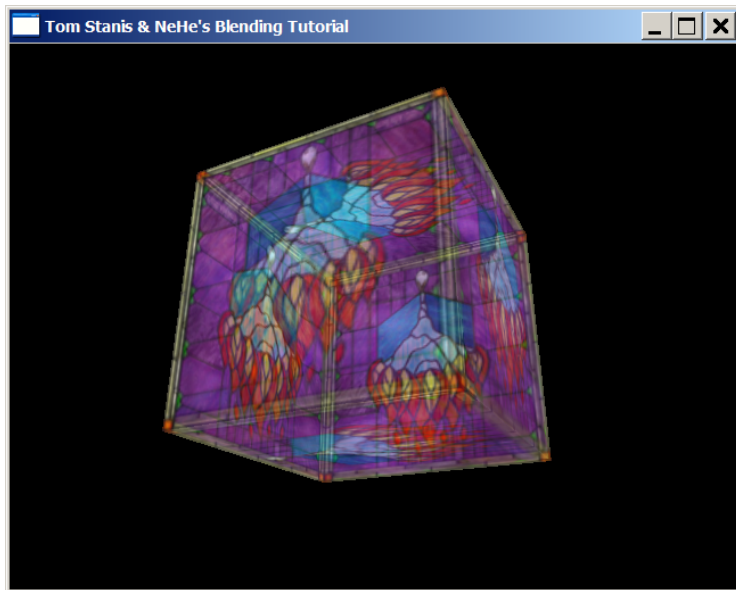
Podstawowe teksturowanie

- 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() ;
```



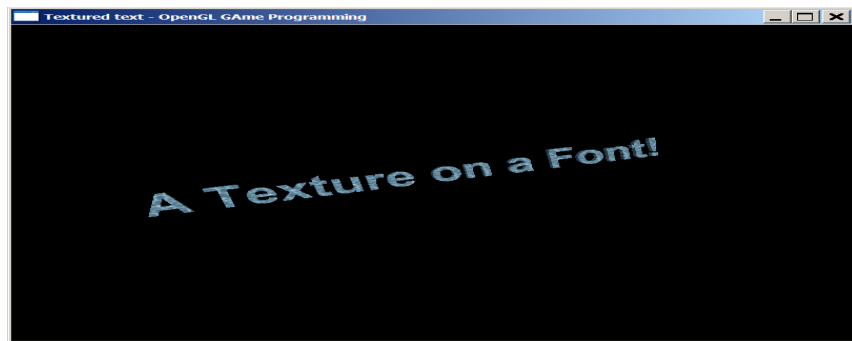
Tekstutowanie z przezroczystością



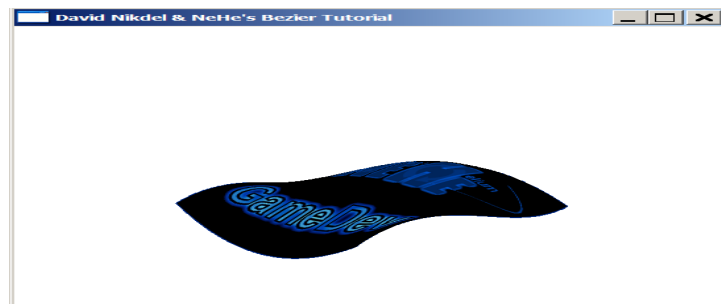
```
//...  
glDepthFunc(GL_LEQUAL);  
glBlendFunc(GL_SRC_ALPHA, GL_ONE);  
//...  
if(blend)  
{ glEnable(GL_BLEND);  
  
    glDisable(GL_DEPTH_TEST);  
}
```

Możliwości teksturowania w OpenGL

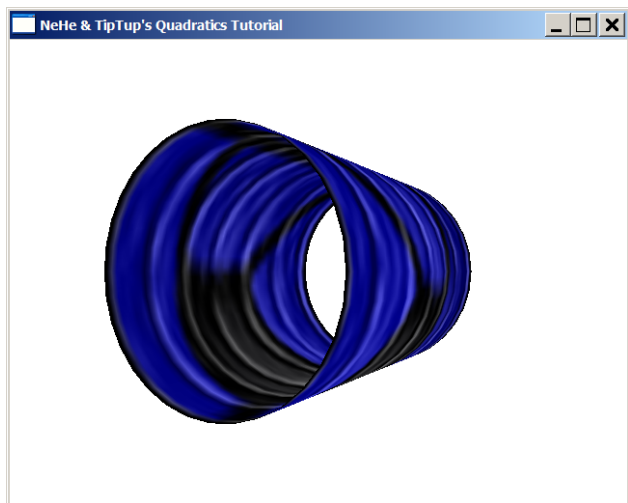
Fonts:



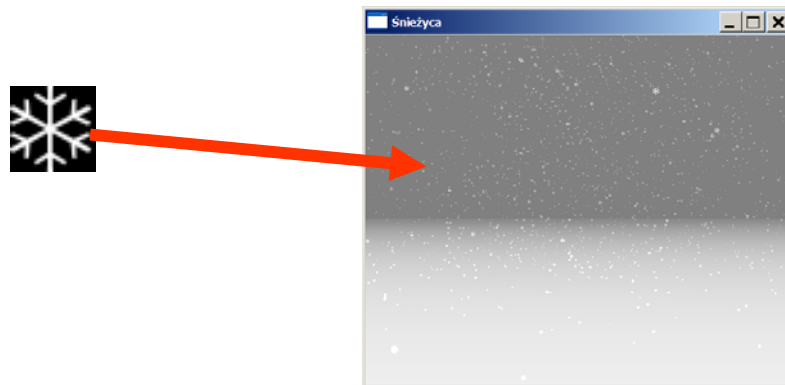
OpenGL surfaces (powierzchnie):



OpenGL quadrics (Kwadryki):



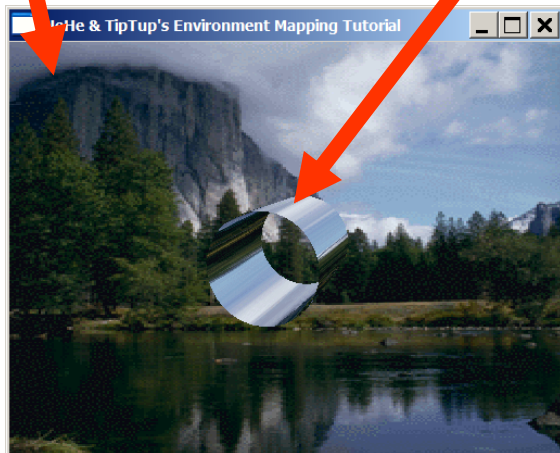
Particles (cząsteczki):



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

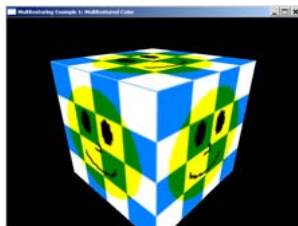
- Potrzebujemy 2 tekstur:



+



->

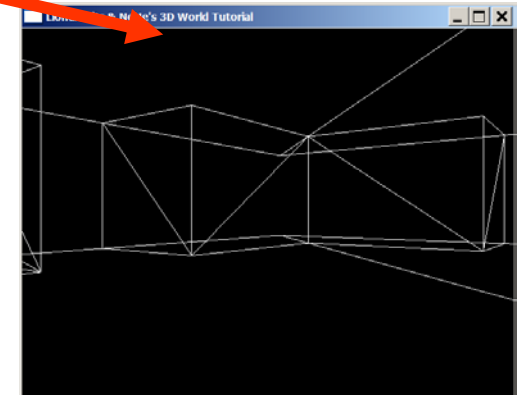
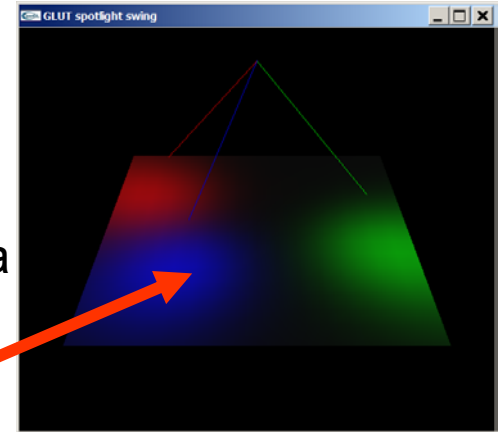


- W OpenGL musi działać rozszerzenie GL_ARB_multitexture:**

```
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 ocienioną 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
- **SPRYTNE ZASTOSOWANIE MULTITEKSTUROWANIA 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.

