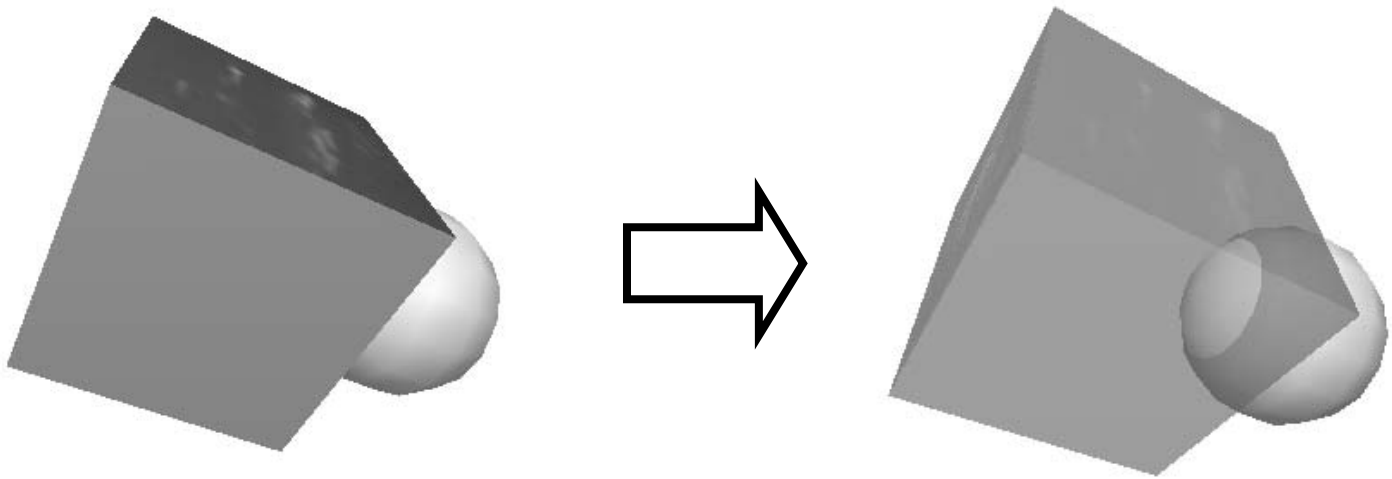
# OpenGL – przezroczystość

- **W standardzie OpenGL efekty przezroczystości uzyskuje się poprzez zezwolenie na łączenie kolorów:**
  - **Kolor piksela tworzy się na podstawie kolorów obiektu przesłanianego i przesłaniającego**
  - **O stopniu przesłonienia decyduje współczynnik Alpha w funkcji ustalania kolorów:**

    **glColor4d(R,G,B,Alpha);**

    **Im niższa wartość współczynnika, tym większa jest przezroczystość danego obiektu.**

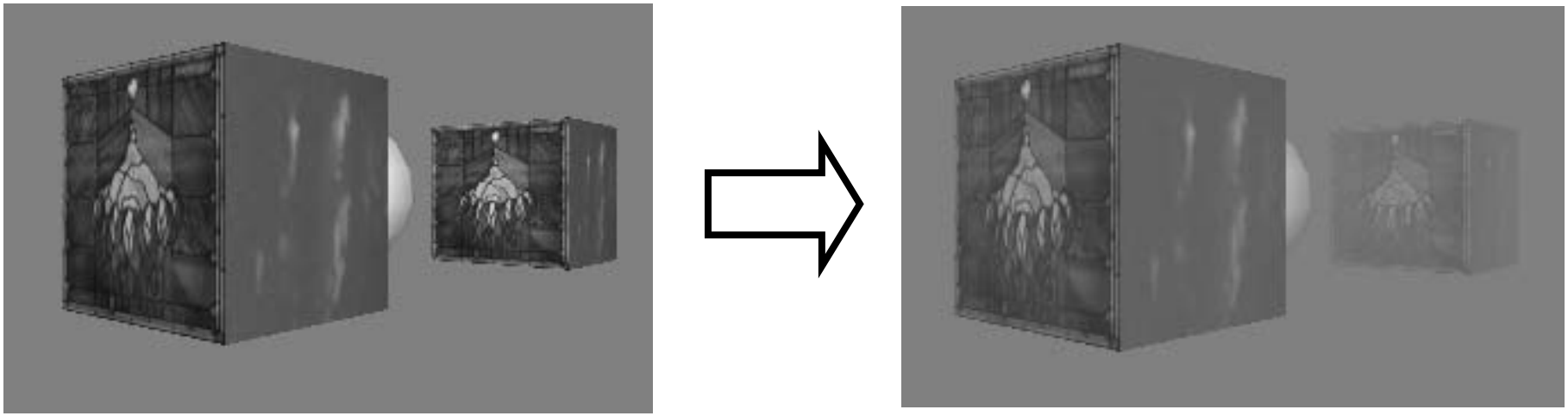


# OpenGL – włączenie przezroczystości

```
{
glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
//Wyrysowanie wszystkich obiektów nieprzezroczystych:
glPushMatrix();
      glTranslated(0,0,-40);
      kula();
glPopMatrix();
//Włączenie łączenia kolorów:
glEnable(GL_BLEND);
//Ustawienie bufora głębokości w tryb odczytu:
glDepthMask(GL_FALSE);
//Skonfigurowanie sposobu łączenia kolorów:
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
//Określenie koloru obiektu przezroczystego:
glColor4d(0.6,0.7,0.6,0.4);
skrzynka();
//Przywrócenie bufora głębokości w pełny tryb zapisu/odczytu:
glDepthMask(GL_TRUE);
//Wyłączenie łączenia kolorów:
glDisable(GL_BLEND);}
```

# OpenGL – mgła

- **W standardzie OpenGL wprowadzono możliwość zdefiniowania mgły.**
- **Stosuje się ją zwykle w dwu przypadkach:**
  - **Do naśladowania rzeczywistego zjawiska mgły,**
  - **Do przesłaniania lub zacierania obiektów znajdujących się dalej na scenie (bardziej realistyczna dal, czy noc),**

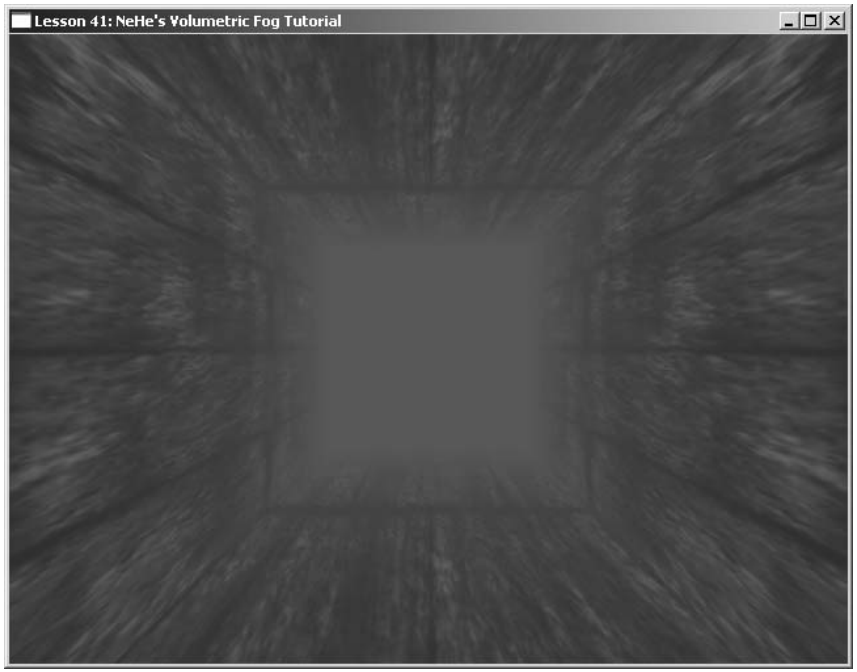


# OpenGL – definiowanie mgły

```
{float fogColor[4]={0.5,0.5,0.5,1.0}; // Kolor mgły

glFogi(GL_FOG_MODE, GL_LINEAR);
// Typ odwzorowania mgły
// GL_LINEAR – (koniec – z) / (koniec – początek)
// GL_EXP – e^(gęstość*głębokość)
// GL_EXP2 - e^(gęstość*głębokość)*(gęstość*głębokość)


glFogfv(GL_FOG_COLOR, fogColor);  // Kolor mgły

glFogf(GL_FOG_DENSITY, 0.0001f);  // Gęstość

glFogf(GL_FOG_START, 50.0f);      // Początek
glFogf(GL_FOG_END, 300.0f);       // Koniec

glEnable(GL_FOG);                 // Włączenie mgły
//...
// Instrukcje rysujące obiekty na scenie
}
```

# OpenGL – mgła objętościowa

- **Zastosowanie standardowej mgły OpenGL przewiduje obliczanie efektu koloru mgły na podstawie tylko odległości względem osi z. W przypadku „rozciągniętych” wzdłuż innych osi obiektów znajdujących się blisko obserwatora, nie następuje „naturalne” przesłonięcie.**
- **Alternatywnym sposobem reprezentowania mgły jest opisywanie go jako oddzielnego bytu reprezentowanego przez sfery, cząsteczki lub mapy mgły.**

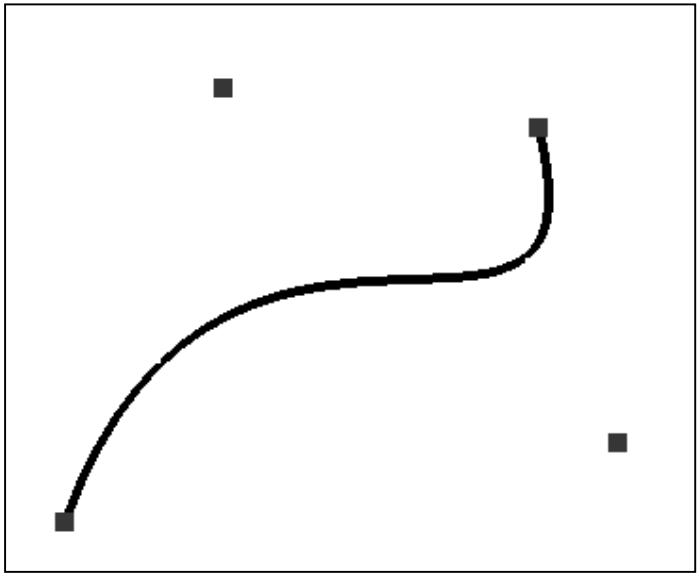



# OpenGL – krzywe parametryczne

- **W standardzie OpenGL przewidziano możliwość konstruowania krzywych parametrycznych (krzywe Beziera i Krzywe NURBS)**
- **Ogólny wzór na krzywe parametryczne: C(u)=[x(u), y(u)]**
- **Krzywe parametryczne opisuje się przez zestaw punktów kontrolnych, które wpływają na ich wygląd:**
  - **W przypadku krzywych Beziera pierwszy i ostatni punkt kontrolny definiują położenie początku i końca krzywej, pozostałe zaś „odginają" krzywą. Zwykle dobre efekty uzyskuje się podając od 3 do 4 punktów kontrolnych, jeśli chce się uzyskać krzywą gładką. Bardziej złożone krzywe definiuje się „sklejając" odcinki opisywane 3-4 punktami kontrolnymi.**
  - **Krzywe NURBS mają właściwości podobne do krzywych Beziera. Ich przewaga polega na tym, że z definicji podzielone są na segmenty. Na własność każdego z segmentów mają wpływ tylko 4 najbliższe punkty kontrolne. (Można łatwo zapewnić gładkość krzywej).**

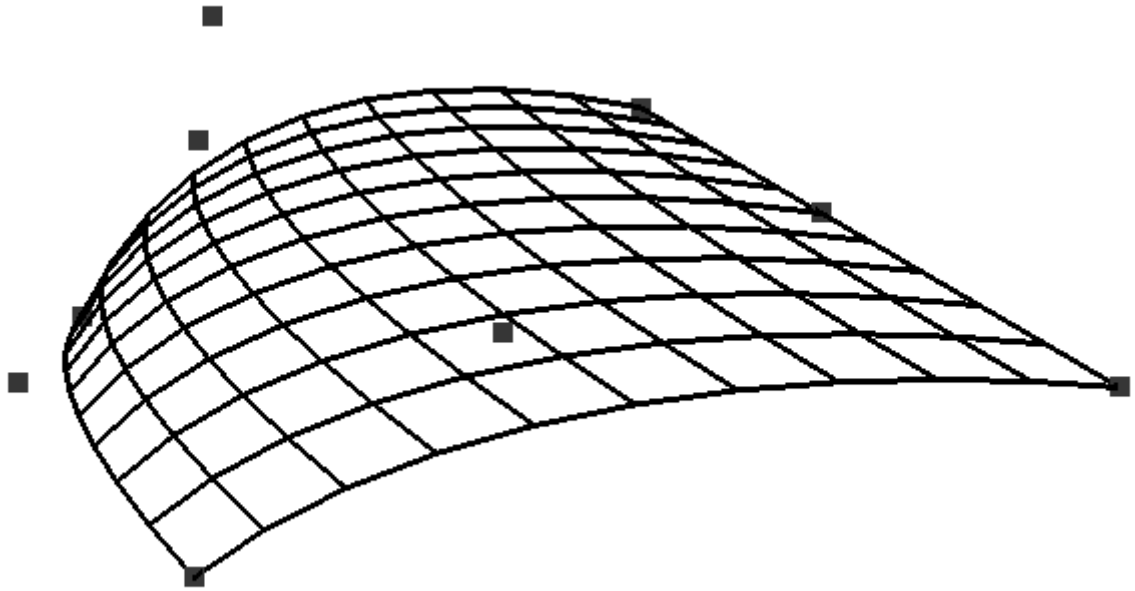
## Definiowanie odcinaka krzywej Beziera (zastosowanie ewaluatora)

```
{// 4 punkty kontrolne:
float control[4][3]={{ 1.0, -3.0, 0.0 }, { 3.0, 2.5, 0.0 },
                     { 8.0, -2.0, 0.0}, {7.0, 2.0, 0.0 } };
glMap1f(GL_MAP1_VERTEX_3, // Jednowymiarowy ewaluator
        0.0,              // początkowa wartość par. u
        30.0,             // końcowa wartość par. u
        3,                // odległość między punkt. krzywej
        4,                // ilość punktów kontrolnych
        &control[0][0]);  // dane punktów kontrolnych
glEnable(GL_MAP1_VERTEX_3);// Aktywowanie formatu pkt. kontr.
// Wyrysowanie krzywej na podstawie ewaluatora:
glMapGrid1d(30,0.0,30.0);
glEvalMesh1(GL_LINE,0,30);




glColor3f(1.0, 0.0, 0.0);
// Wyrysowanie punktów kontrolnych:
glBegin(GL_POINTS);
for (i = 0; i < 4; i++)
glVertex3fv(&control[i][0]); glEnd();}
```



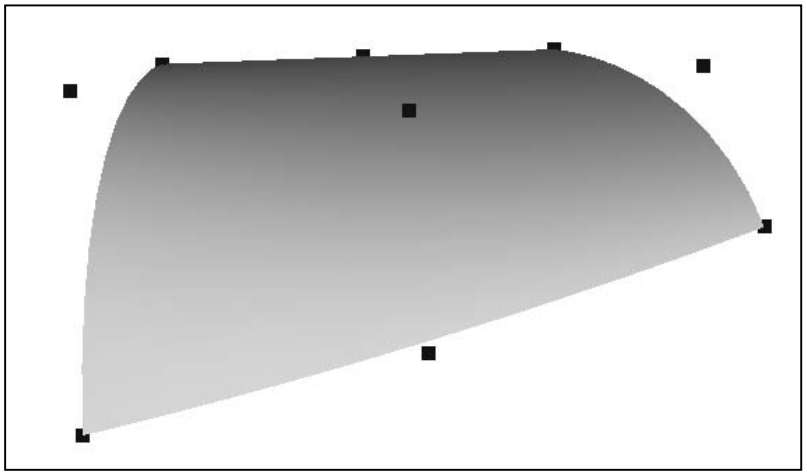
# OpenGL – powierzchnie parametryczne

- **Powierzchnie parametryczne są rozwinięciem pojęcia krzywych parametrycznych.**
- **Ogólny wzór na powierzchnie parametryczne: S(u,v)=[x(u,v), y(u,v), z(u,v)]**
- **Punkty kontrolne są odpowiedzialne teraz za „wyginanie” powierzchni.**



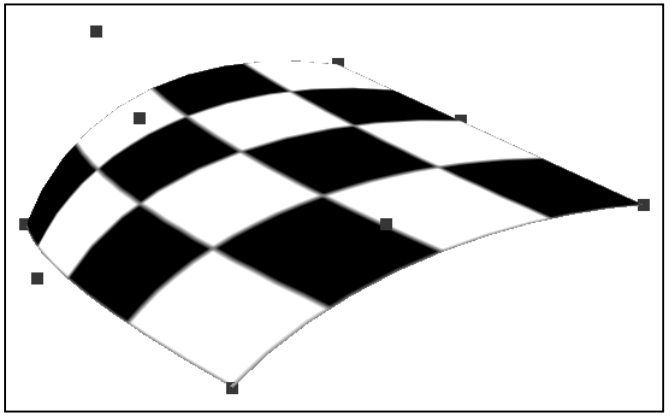
# Definiowanie powierzchni Beziera

```
{//punkty kontrolne:
float cSurface[3][3][3] = {
{{-200.0,40.0,200.0}, {-100.0,100.0,200.0},{200.0,0.0,200.0}},
{{-240.0,0.0,0.0},{-150.0,100.0,0.0},{200.0,0.0,0.0 }},
{{-200.0,-80.0,-200.0},{-100.0,100.0,-200.0},{200.0,0.0,-200.0
}}};
// Utworzenie powierzchni Beziera i jej aktywacja
glMap2f(GL_MAP2_VERTEX_3,
0.0, 10.0, 3, 3, 0.0, 10.0, 9, 3, &cSurface[0][0][0]);
glEnable(GL_MAP2_VERTEX_3);
// tworzeniw siatki równoodległej:
glMapGrid2f(10, 0.0f, 10.0f, 10, 0.0f, 10.0f);
// Automatyczna generacja normalnych
glEnable(GL_AUTO_NORMAL);
// wyznaczenie powierzchni
// na podstawie siatki
glEvalMesh2(GL_FILL, 0, 10, 0, 10);
glBegin(GL_POINTS);
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
glVertex3fv(&cSurface[i][j][0]);
glEnd();}
```



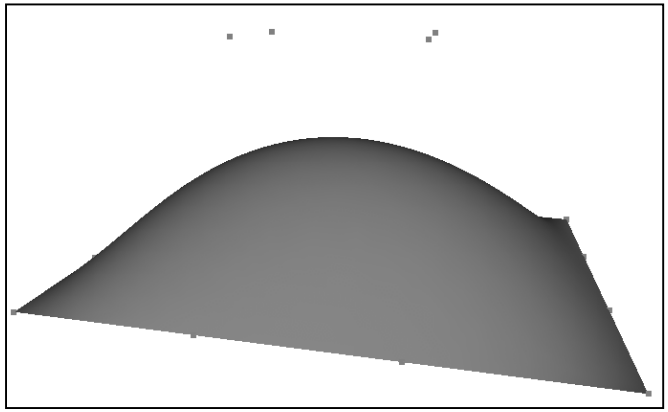
# Nakładanie tekstury na powierzchnię Beziera

```
{// punkty kontrolne krzywej i tekstury
float cSurface[3][3][3] = {
{{-200.0,40.0,200.0},{-100.0,100.0,200.0},{200.0,0.0,200.0}},
{{-240.0, 0.0, 0.0},{-150.0,100.0,0.0},{200.0,0.0,0.0}},
{{-200.0,-80.0,-200.0},{-100.0,100.0,-200.0},{200.0,0.0,-200.0
}}};
float sTexCoords[2][2][2] = {{{0.0,0.0},{0.0,1.0}},
                             {{1.0,0.0},{1.0,1.0}}};

// tworzy ewaluator punktów kontrolnych powierzchni
glMap2f(GL_MAP2_VERTEX_3, 0.0, 1.0, 3, 3, 0.0, 1.0, 9, 3,
      &cSurface[0][0][0]);
// tworzy ewaluator współrzędnych tekstury
glMap2f(GL_MAP2_TEXTURE_COORD_2, 0, 1, 2, 2, 0, 1, 4, 2,
      &sTexCoords[0][0][0]);
// aktywuje utworzone ewaluatory
glEnable(GL_MAP2_TEXTURE_COORD_2);
glEnable(GL_MAP2_VERTEX_3);
// tworzy siatkę powierzchni
glMapGrid2f(10, 0.0f, 1.0f,
            10, 0.0f, 1.0f);
glEvalMesh2(GL_FILL, 0, 10, 0, 10);}
```



# OpenGL – powierzchnie B-sklejane (NURBS)

```
{
GLUnurbsObj *myNurb;
// Punkty kontrolne:
float knots[8] = { 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0 };
float nurb[4][4][3];
// inicjuje obiekt krzywej B-sklejanej
myNurb = gluNewNurbsRenderer();
gluNurbsProperty(myNurb, GLU_SAMPLING_TOLERANCE, 50.0);
gluNurbsProperty(myNurb, GLU_DISPLAY_MODE, GLU_FILL);
// rozpoczyna definiowanie powierzchni B-sklejanej
gluBeginSurface(myNurb);
// rysuje powierzchnię B-sklejaną
gluNurbsSurface(myNurb, 8, knots, 8, knots, 4*3, 3,
&nurb[0][0][0], 4, 4, GL_MAP2_VERTEX_3);
// skończone
gluEndSurface(myNurb);}
```

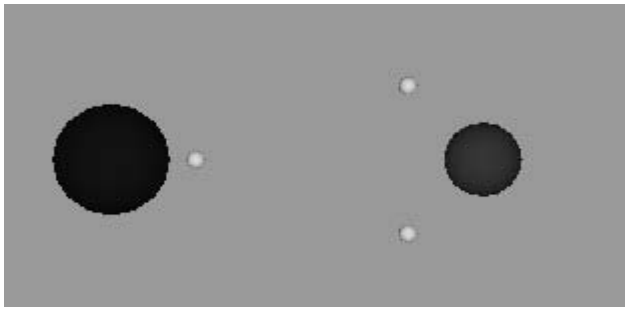


# Podsumowanie zasad stosowania OpenGL do tworzenia scen

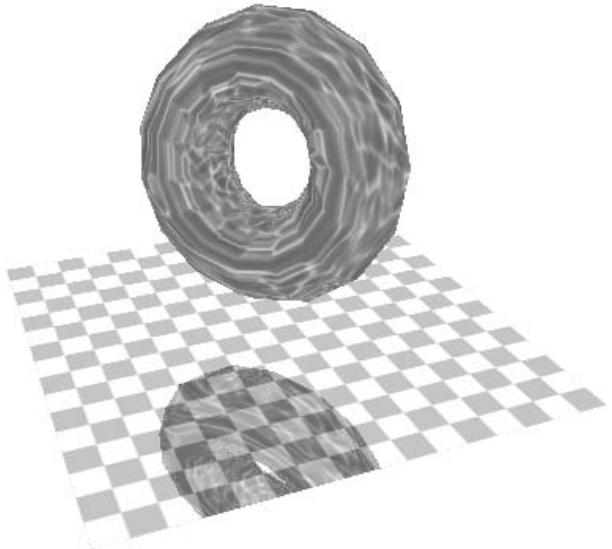
1. **Przygotowanie siatek**
2. **Zdefiniowanie transformacji przestrzennych i położenia kamery**
3. **Zdefiniowanie oświetlenia sceny**
4. **Teksturowanie**
5. **Zdefiniowanie interfejsu użytkownika**
6. **Dodatkowe elementy – mgła, przezroczystość, mapy oświetlenia itp.**

# Przegląd technik grafiki komputerowej

- **Selekcja elementów sceny (OpenGL):**

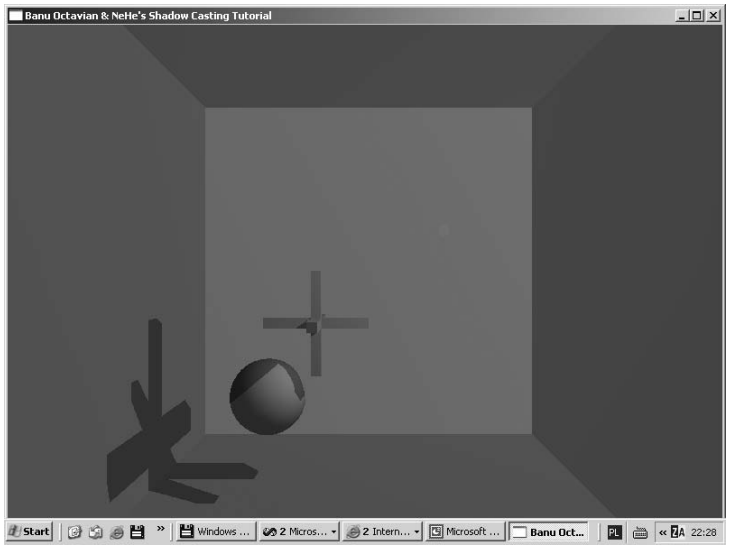


- **Zastosowanie bufora powielania do tworzenia realistycznego efektu odbicia (OpenGL):**



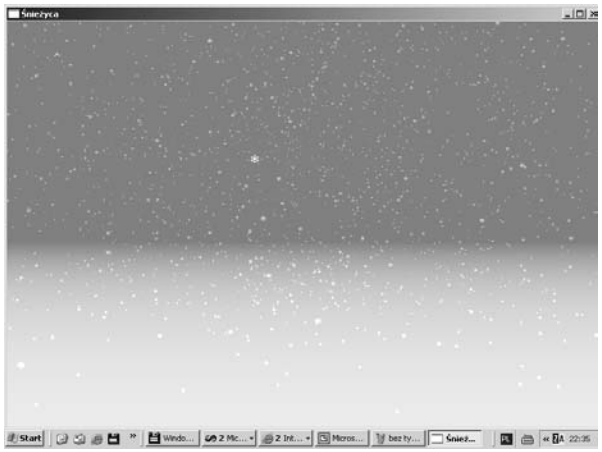
# Przegląd technik grafiki komputerowej

- **Rysowanie cieni (OpenGL):**




- **Systemy cząsteczek:**

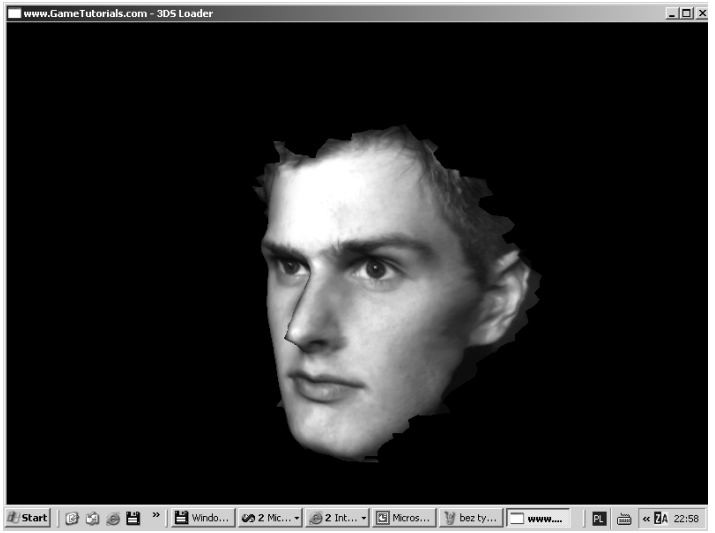
**silnik rakietowy**



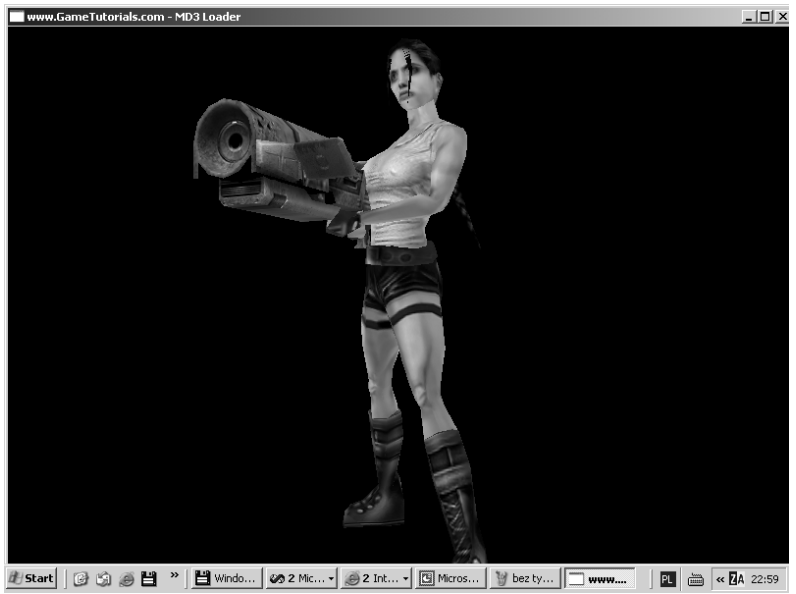
**śnieżyca**

# Przegląd technik grafiki komputerowej

- **Ładowanie obiektów 3DS**

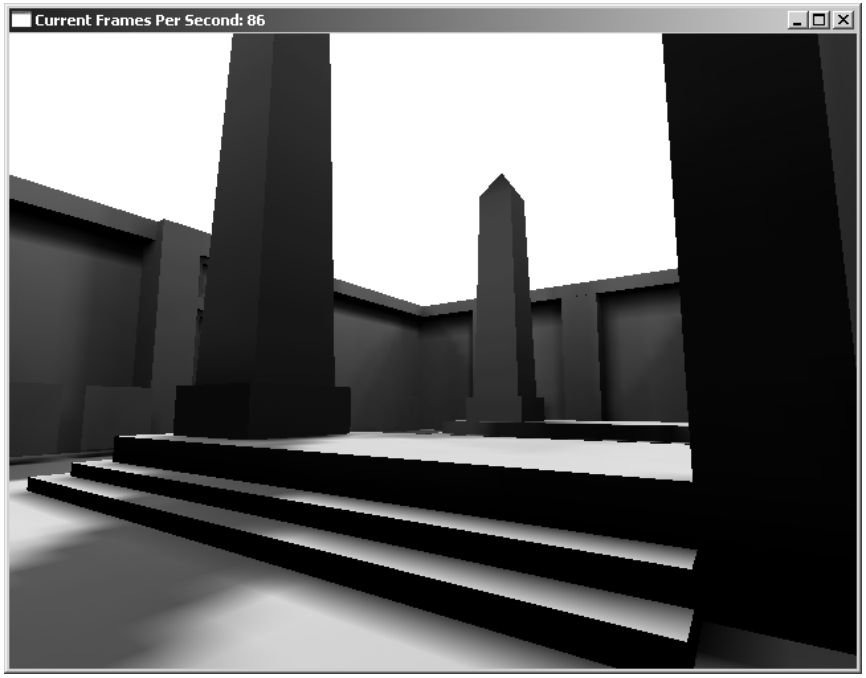



- **Ładowanie plików MD3**



# Przegląd technik grafiki komputerowej

- **Ładowanie plików BSP**




- **Zastosowanie modeli fizycznych**