5 Kamera i interakcija

U ovom poglavlju biće objašnjen rad sa kamerom kao i interakcija sa korisnikom.

5.1 Rad sa kamerom

Za specifikovanje pozicije kamere koristi se *Viewing* matrica, videti poglavlje o transformacijama. Pozicioniranje kamere se realizuje korišćenjem metode **gluLookAt**.

```
public static void gluLookAt(
    double eyex, double eyey, double eyez,
    double centerx, double centery,  double centerz,
    double upx, double upy, double upz)
```

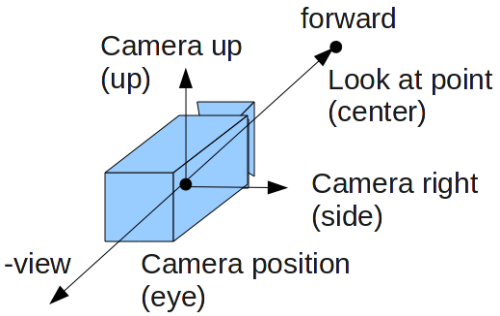
gde su:

`eyex, eyey, eyez` – tačka posmatranja,

`centerx, centery, centerz` – vektor koji opisuje tačku u koju kamera gleda,

`upx, upy, upz` – vektor koji određuje pravac i smer na gore (*upward vector*).

Pozicija i orijentacija objekta (obično se koristi termin *akter*), a i kamere, u prostoru se jednoznačno opisuje pomoću tri atributa: pozicije objekta, vektora koji definiše tačku ka kojoj je objekat orijentisan (za vrednost ove tačke može se uzeti bilo koja vrednost sa putanje pozicije kamere ka *center* poziciji) i vektora koji definiše šta je iznad (*upward vector*), slika 5.1.1. Na ovaj način se značajno olakšavaju transformacije nad objektima. Olakšane su transformacije objekata u odnosu na druge objekte, npr. primer revolucije Zemlje i Meseca oko Sunca – Mesec rotira u odnosu na Zemlju i sa njom oko Sunca. Lokalni koordinatni sistem objekta određen na ovaj način se naziva *frame of reference*.



Slika 5.1.1 Jednoznačno određivanje pozicije i orijentacije objekta u prostoru

5.2 Interakcija korisnika sa tastaturom

Budući da koristimo SharpGL kontrolu unutar WPF tehnologije, samo registrovanje korisničkog unosa biće realizovano preko implementacije handlera za događaje koji su nam od interesa. U slučaju unosa sa tastature, implementiramo obrađivač za OnKeyDown event. Ovaj događaj obrađujemo na nivou celog glavnog prozora aplikacije, primer u kojem se preko obrađivača modifikuje ugao rotacije dat je u listingu 5.2.1.

```
        private void Window_KeyDown(object sender, KeyEventArgs e)
        {
            switch (e.Key)
            {
                case Key.A: m_world.Angle -= 5.0f; break;
                case Key.D: m_world.Angle += 5.0f; break;
            }
        }
```

Listing 5.2.1 Primer interakcije sa korisnikom – obrađivač pritiska tipke tastature

5.3 Interakcija korisnika sa mišem

Prilikom rada sa mišem, definišemo obrađivače za OnMouseMove event. U Listingu 5.3.1. dat je primer obrađivača događaja koji ažurira rotaciju kamere iz prvog lica tako što šalje pomeraj kursora u odnosu na prethodno pozivanje istog obrađivača. Ova funkcija će takođe detektovati ukoliko korisnik pokuša da napusti prozor sa kursorom i vratiće ga na sredinu.

```
        private void Window_MouseMove(object sender, MouseEventArgs e)
        {
            bool outOfBoundsX = false;
            bool outOfBoundsY = false;
            Point point = e.GetPosition(this);

            if (point.Y <= BORDER || point.Y >= this.Height - BORDER)
            {
                outOfBoundsY = true;
            }
            if (point.X <= BORDER || point.X >= this.Width - BORDER)
            {
                outOfBoundsX = true;
            }

            if (!outOfBoundsY && !outOfBoundsX)
            {
                double deltaX = oldPos.X - point.X;
                double deltaY = oldPos.Y - point.Y;
                m_world.UpdateCameraRotation(deltaX, deltaY);
                oldPos = point;
            }
```

```
            else
            {
                if (outOfBoundsX)
                {
                    SetCursorPos((int)this.Left + (int)this.Width / 2,
(int)this.Top + (int)point.Y);
                    oldPos.X = this.Width / 2;
                    oldPos.Y = point.Y;
                }
                else
                {
                    SetCursorPos((int)this.Left     +     (int)point.X,
(int)this.Top + (int)this.Height / 2);
                    oldPos.Y = this.Height / 2;
                    oldPos.X = point.X;
                }
            }
}
```

Listing 5.3.1 Primer interakcije sa korisnikom – obrađivač pritiska

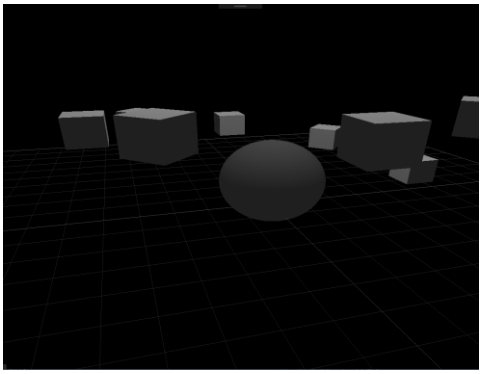
5.4 Primer upravljanja kamerom pomoću tastature i miša

U nastavku teksta biće prezentovan primer upravljanja kamerom korišćenjem W,A,S,D tastera, kao i pomeranjem miša za rotaciju pogleda. Programski kod primera prikazan je u listingu 5.4.1. Rezultat rada aplikacije je prikazan na slici 5.4.1. Klasa koju koristimo da podešavamo parametre kamere je LookAtCamera. Ova klasa nije ništa drugo do set atributa koji opisuju poziciju i orijentaciju kamere, kao i transformaciju projekcije preko glLookAt metode. Ideja za rotiranje preko pomeraja kursora počiva na kreiranju jedinične sfere oko kamere i zatim postavljanja Target atributa kamere na tačku koja se nalazi na toj sferi. Više informacija o sfernom koordinatnom sistemu i konverziji koordinata u Dekartov trodimenzionalni koordinatni sistem može se naći u literaturi.

```
        /// <summary>
        ///  Azurira rotaciju kamere preko pomeraja misa
        /// </summary>
        public void UpdateCameraRotation(double deltaX, double deltaY)
        {
            horizontalAngle += mouseSpeed * deltaX;
            verticalAngle += mouseSpeed * deltaY;

            direction.X = (float)(Math.Cos(verticalAngle) *
Math.Sin(horizontalAngle));
            direction.Y = (float)(Math.Sin(verticalAngle));
            direction.Z = (float)(Math.Cos(verticalAngle) *
Math.Cos(horizontalAngle));

            right.X = (float)Math.Sin(horizontalAngle - (Math.PI /
2));
```

```
            right.Y = 0f;
            right.Z = (float)Math.Cos(horizontalAngle - (Math.PI /
2));

            up = right.VectorProduct(direction);

            lookAtCam.Target = lookAtCam.Position + direction;
            lookAtCam.UpVector = up;
        }

        /// <summary>
        ///  Azurira poziciju kamere preko tipki tastature
        /// </summary>
        public void UpdateCameraPosition(int deltaX, int deltaY, int
deltaZ)
        {
            Vertex deltaForward = direction * deltaZ;
            Vertex deltaStrafe = right * deltaX;
            Vertex deltaUp = up * deltaY;
            Vertex delta = deltaForward + deltaStrafe + deltaUp;
            lookAtCam.Position += (delta * walkSpeed);
            lookAtCam.Target = lookAtCam.Position + direction;
            lookAtCam.UpVector = up;
        }
```

Listing 5.4.1 Primer interakcije sa korisnikom – upravljanje kamerom



Slika 5.4.1 Rezultat primera iz listinga 5.4.1

6 Boja i senčenje objekata

OpenGL koristi RGB model boja za specifikovanje boje. Svaka od komponenti se opisuje jednim bajtom koji definiše intenzitet te komponente. Koristi se i RGBA model boja, koji predstavlja RGB model proširen *Alpha* komponentom koja definiše prozirnost (*transparency*).

U OpenGL standardu boja se definiše pomoću familije metoda **glColor**. Najčešće se koriste sledeće metode:

```
public static void glColor3f(float R, float G,
                             float B)
public static void glColor4f(float R, float G,
                             float B, float alpha)
public static void glColor3ub(byte R,byte G,byte B)
```

gde su:

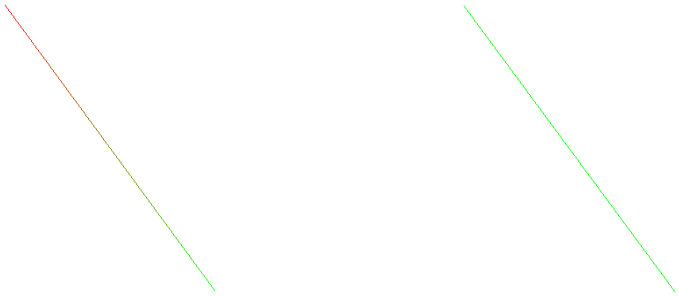
`R,G,B,alpha` – RGBA komponente boje, kod metoda sa parametrima *float* tipa podataka vrednosti su normalizovane [0.0f, 1.0f], dok *byte* verzije su u opsegu [0, 255]. Metode koje nemaju *alpha* kao parametar podrazumevaju *alpha* jednako 1.0f, odnosno 255.

Svi objekti koji iscrtavaju posle poziva *glColor* metode biće iscrtani u boji koju je taj poziv specifikovao. Boja se uvek odnosi na tačku – teme objekta. Tako da ako su dva temena nekog objekta obojena različitim bojama, postavlja se pitanje koje će boje biti linija koja ih povezuje, odnosno kojom bojom će biti ispunjen poligon kojem temena pripadaju. Ovo direktno zavisi od tipa senčenja.

OpenGL podržava dva tipa senčenja: **flat** i **smooth**. Ako u sceni ne postoji osvetljenje *flat* tip senčenja (homogeno (*Lambert*) senčenje) će liniju između dva temena obojiti bojom drugog temena, dok *smooth* senčenje (*Gouraud* senčenje) će obojiti liniju prelivom boja temena. Slično se realizuje senčenje poligona, sa razlikom da u definisanju senčenja učestvuje više temena. Slike 6.1 i 6.2 prikazuju tipove senčenja primenjene na poligon, odnosno liniju.

a) *smooth* tip senčenja b) *flat* tip senčenja

Slika 6.1 Tipovi senčenja na primeru trougla



a) *smooth* tip senčenja b) *flat* tip senčenja

Slika 6.2 Tipovi senčenja na primeru linije

Model senčenja se definiše pomoću metode **glShadeModel**, koja ima oblik:

```
public static void glShadeModel(int mode)
```

gde je:

`mode` – tip senčenja objekta. Dozvoljene vrednosti su:*GL_SMOOTH* i *GL_FLAT*. Podrazumevana vrednost je *GL_SMOOTH.*

Bitno je naglasiti da kod senčenja poligona korišćenjem flat tipa senčenja, boja ispunjavanja celog poligona je poslednja definisana boja temena.

7 Materijali

U OpenGL standardu materijal se opisuje preko svojih reflektivnih karakteristika. Materijal se opisuje preko tri svetlosne komponente: ambijentalne, difuzne i spekularne.

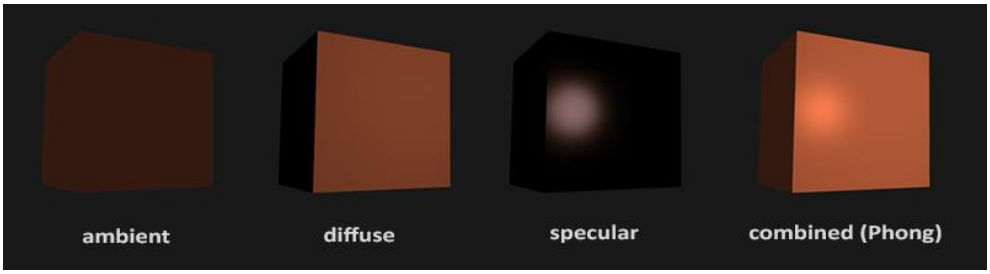
Ambijentalno svetlo osvetljava objekte jednako i „kao“ da je sveprisutno tj. ne dolazi iz nekog konkretnog pravca. Primer ovog osvetljenja je dnevna svetlost.

Difuzno svetlo osvetljava objekte iz nekog konkretnog pravca. Svetlosni zraci se prelamaju i rasipaju na površini objekta. Ova komponenta svetla najintenzivnije obasjava poligone čija je normala uperena u poziciju svetla. Primer ovog osvetljenja je stona lampa.

Spekularno svetlo (*specular highlight*), slično kao i difuzno svetlo, dolazi iz nekog pravca, ali refleksija je pod mnogo oštrijim uglom i nema rasipanja. Ova svetlost kreira veoma jako osvetljene površine na objektu – odsjaj.

Pored ovih osnovnih komponenti može se definisati i **emisiona** komponenta. Ona predstavlja boju koju materijal isijava.

Slika 7.1 prikazuje kocku kojoj su definisane ambijentalna, difuzna i spekularna komponenta materijala.



Slika 7.1 Osnovne tri komponente Phong osvetljenja

Materijal se definiše pomoću familije metoda **glMaterial**, od kojih se najčešće koristi:

```
public static void glMaterialfv(int face,int pname,
                                 IntPtr params)
public static void glMaterialfv(int face,int pname,
                                 float[] params)
```

gde su:

`face` – tip orijentacije (tip poligona, videti poglavlje vezano za orijentaciju poligona) nad kojim se primenjuje materijal. Dozvoljene vrednosti su: *GL_FRONT*, *GL_BACK* ili *GL_FRONT_AND_BACK.* Početna vrednost je *GL_FRONT_AND_BACK*,

`pname` – koja komponenta se podešava sa *param* parametrom. Moguće vrednosti su: *GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR,*

GL_EMISSION, GL_SHININESS, GL_COLOR_INDEXES, i *GL_AMBIENT_AND_DIFFUSE.* Početna vrednost je *GL_AMBIENT_AND_DIFFUSE,*

`params` – parametar čija semantika zavisi od *pname*, uglavnom je boja.

Ovakav način navođenja materijala za svaki objekat nije uvek pogodan. OpenGL nudi alternativni način definisanja materijala – **color tracking**. *Color tracking* podrazumeva definisanje materijala preko poziva *glColor* metode.

U *color tracking* režim se ulazi pozivom metode *glEnable* sa argumentom *GL_COLOR_MATERIAL*. Funkcijom **glColorMaterial** se specifikuje na koje orijentacije poligona i koje komponente osvetljenja se boja materijala odnosi (parametri ove metode su isti sa prva dva parametra metode *glMaterialfv*). U praksi se najčešće koristi ovakav način definisanja materijala.

```
public static void glColorMaterial(int face,
                                   int mode)
```

gde su:

`face` – tip orijentacije (tip poligona, videti poglavlje vezano za orijentaciju poligona) nad kojim se primenjuje materijal. Dozvoljene vrednosti su: *GL_FRONT*, *GL_BACK* ili *GL_FRONT_AND_BACK.* Početna vrednost je *GL_FRONT_AND_BACK*,

`mode` – koja komponenta se podešava sa *param* parametrom. Moguće vrednosti su: *GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_AMBIENT_AND_DIFFUSE,* i *GL_EMISSION.* Početna vrednost je *GL_AMBIENT_AND_DIFFUSE.*

Listing 7.1 prikazuje primer kocke i piramide kojima su pridruženi materijali korišćenjem *color tracking* tehnike.

```
using SharpGL;
using SharpGL.SceneGraph.Primitives;
using SharpGL.Enumerations;


namespace Materials
{
    class World
    {
        #region Atributi

        float triangleRotation = 0;
        float quadRotation = 0;
        private ShadeModel m_selectedModel;

        #endregion
```

```
        #region Konstruktori

        /// <summary>
        ///          Konstruktor opengl sveta.
        /// </summary>
        public World()
        {
            m_selectedModel = ShadeModel.Smooth;
        }

        #endregion

        #region Metode

        /// <summary>
        /// Korisnicka inicijalizacija i podesavanje OpenGL parametara
        /// </summary>
        public void Initialize(OpenGL gl)
        {
            float[] whiteLight = { 1.0f, 1.0f, 1.0f, 1.0f };

            gl.Enable(OpenGL.GL_DEPTH_TEST);
            gl.Enable(OpenGL.GL_CULL_FACE);
            gl.Enable(OpenGL.GL_COLOR_MATERIAL);
            gl.ColorMaterial(OpenGL.GL_FRONT, OpenGL.GL_AMBIENT);
            gl.LightModel(LightModelParameter.Ambient, whiteLight);
            gl.ClearColor(0f, 0f, 0f, 1.0f);
        }

        /// <summary>
        /// Podesava viewport i projekciju za OpenGL kontrolu.
        /// </summary>
        public void Resize(OpenGL gl, int width, int height)
        {
            gl.MatrixMode(OpenGL.GL_PROJECTION);
            gl.LoadIdentity();
            gl.Perspective(45f, (double)width / height, 0.1f, 500f);
            gl.MatrixMode(OpenGL.GL_MODELVIEW);
        }

        /// <summary>
        ///  Iscrtavanje OpenGL kontrole.
        /// </summary>
        public void Draw(OpenGL gl)
        {
            gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT |
OpenGL.GL_DEPTH_BUFFER_BIT);       // Clear The Screen And The Depth
Buffer
```

```
            gl.LoadIdentity();                                  // Reset
The View

            DrawGrid(gl);

            gl.Translate(-1.5f, 0.0f, -6.0f);               // Move
Left And Into The Screen

            gl.Rotate(triangleRotation, 0.0f, 1.0f, 0.0f);
// Rotate The Pyramid On It's Y Axis

            gl.Begin(OpenGL.GL_TRIANGLES);                  // Start
Drawing The Pyramid

            gl.Color(1.0f, 0.0f, 0.0f);         // Red
            gl.Vertex(0.0f, 1.0f, 0.0f);            // Top Of Triangle
(Front)
            gl.Color(0.0f, 1.0f, 0.0f);         // Green
            gl.Vertex(-1.0f, -1.0f, 1.0f);          // Left Of
Triangle (Front)
            gl.Color(0.0f, 0.0f, 1.0f);         // Blue
            gl.Vertex(1.0f, -1.0f, 1.0f);           // Right Of
Triangle (Front)

            gl.Color(1.0f, 0.0f, 0.0f);         // Red
            gl.Vertex(0.0f, 1.0f, 0.0f);            // Top Of Triangle
(Right)
            gl.Color(0.0f, 0.0f, 1.0f);         // Blue
            gl.Vertex(1.0f, -1.0f, 1.0f);           // Left Of
Triangle (Right)
            gl.Color(0.0f, 1.0f, 0.0f);         // Green
            gl.Vertex(1.0f, -1.0f, -1.0f);          // Right Of
Triangle (Right)

            gl.Color(1.0f, 0.0f, 0.0f);         // Red
            gl.Vertex(0.0f, 1.0f, 0.0f);            // Top Of Triangle
(Back)
            gl.Color(0.0f, 1.0f, 0.0f);         // Green
            gl.Vertex(1.0f, -1.0f, -1.0f);          // Left Of
Triangle (Back)
            gl.Color(0.0f, 0.0f, 1.0f);         // Blue
            gl.Vertex(-1.0f, -1.0f, -1.0f);         // Right Of
Triangle (Back)

            gl.Color(1.0f, 0.0f, 0.0f);         // Red
            gl.Vertex(0.0f, 1.0f, 0.0f);            // Top Of Triangle
(Left)
            gl.Color(0.0f, 0.0f, 1.0f);         // Blue
            gl.Vertex(-1.0f, -1.0f, -1.0f);         // Left Of
Triangle (Left)
```

```
            gl.Color(0.0f, 1.0f, 0.0f);          // Green
            gl.Vertex(-1.0f, -1.0f, 1.0f);          // Right Of
Triangle (Left)
            gl.End();                        // Done Drawing The
Pyramid

            gl.LoadIdentity();
            gl.Translate(1.5f, 0.0f, -7.0f);                // Move
Right And Into The Screen

            gl.Rotate(quadRotation, 1.0f, 1.0f, 1.0f);          //
Rotate The Cube On X, Y & Z

            gl.Begin(OpenGL.GL_QUADS);                  // Start
Drawing The Cube

            gl.Color(0.0f, 1.0f, 0.0f);          // Set The Color To
Green
            gl.Vertex(1.0f, 1.0f, -1.0f);           // Top Right Of
The Quad (Top)
            gl.Vertex(-1.0f, 1.0f, -1.0f);          // Top Left Of The
Quad (Top)
            gl.Vertex(-1.0f, 1.0f, 1.0f);           // Bottom Left Of
The Quad (Top)
            gl.Vertex(1.0f, 1.0f, 1.0f);            // Bottom Right Of
The Quad (Top)

            gl.Color(1.0f, 0.5f, 0.0f);          // Set The Color To
Orange
            gl.Vertex(1.0f, -1.0f, 1.0f);           // Top Right Of
The Quad (Bottom)
            gl.Vertex(-1.0f, -1.0f, 1.0f);          // Top Left Of The
Quad (Bottom)
            gl.Vertex(-1.0f, -1.0f, -1.0f);         // Bottom Left Of
The Quad (Bottom)
            gl.Vertex(1.0f, -1.0f, -1.0f);          // Bottom Right Of
The Quad (Bottom)

            gl.Color(1.0f, 0.0f, 0.0f);          // Set The Color To
Red
            gl.Vertex(1.0f, 1.0f, 1.0f);            // Top Right Of
The Quad (Front)
            gl.Vertex(-1.0f, 1.0f, 1.0f);           // Top Left Of The
Quad (Front)
            gl.Vertex(-1.0f, -1.0f, 1.0f);          // Bottom Left Of
The Quad (Front)
            gl.Vertex(1.0f, -1.0f, 1.0f);           // Bottom Right Of
The Quad (Front)
```

```
            gl.Color(1.0f, 1.0f, 0.0f);          // Set The Color To
Yellow
            gl.Vertex(1.0f, -1.0f, -1.0f);           // Bottom Left Of
The Quad (Back)
            gl.Vertex(-1.0f, -1.0f, -1.0f);          // Bottom Right Of
The Quad (Back)
            gl.Vertex(-1.0f, 1.0f, -1.0f);           // Top Right Of
The Quad (Back)
            gl.Vertex(1.0f, 1.0f, -1.0f);            // Top Left Of The
Quad (Back)

            gl.Color(0.0f, 0.0f, 1.0f);          // Set The Color To
Blue
            gl.Vertex(-1.0f, 1.0f, 1.0f);            // Top Right Of
The Quad (Left)
            gl.Vertex(-1.0f, 1.0f, -1.0f);           // Top Left Of The
Quad (Left)
            gl.Vertex(-1.0f, -1.0f, -1.0f);          // Bottom Left Of
The Quad (Left)
            gl.Vertex(-1.0f, -1.0f, 1.0f);           // Bottom Right Of
The Quad (Left)

            gl.Color(1.0f, 0.0f, 1.0f);          // Set The Color To
Violet
            gl.Vertex(1.0f, 1.0f, -1.0f);            // Top Right Of
The Quad (Right)
            gl.Vertex(1.0f, 1.0f, 1.0f);             // Top Left Of The
Quad (Right)
            gl.Color(0.0f, 1.0f, 1.0f);
            gl.Vertex(1.0f, -1.0f, 1.0f);            // Bottom Left Of
The Quad (Right)
            gl.Vertex(1.0f, -1.0f, -1.0f);           // Bottom Right Of
The Quad (Right)
            gl.End();                        // Done Drawing The Q

            gl.Flush();

            triangleRotation += 3.0f;// 0.2f;
              // Increase The Rotation Variable For The Triangle
            quadRotation -= 3.0f;// 0.15f;
              // Decrease The Rotation Variable For The Quad

            gl.Flush();
                                }

}

}
```

Listing 7.1 Primer korišćenja *color tracking* tehnike za definisanje materijala

Listing 7.2 prikazuje primer kupe i sfere kojima su pridruženi materijali korišćenjem *color tracking* tehnike. Ovde koristimo SharpGL klase Sphere i Cylinder koje unutar svojih Material atributa metodom Bind() pozivaju funkcije za definisanje materijala, što se može videti u komentaru koji sadrži source code SharpGL-a.

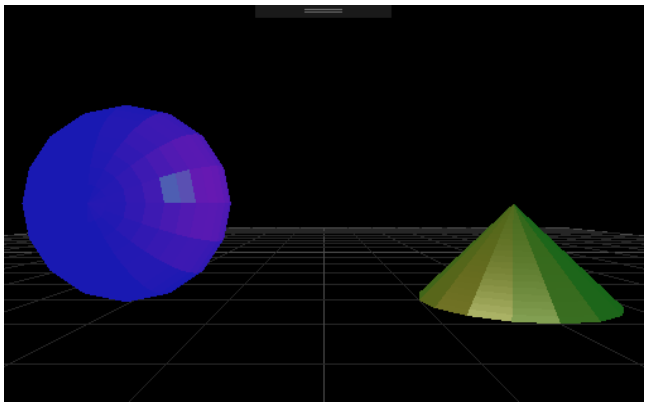
```
using System.Drawing;
using SharpGL;
using SharpGL.SceneGraph.Primitives;
using SharpGL.Enumerations;
using SharpGL.SceneGraph.Quadrics;
using SharpGL.SceneGraph.Assets;

namespace Materials
{
    class World
    {
        #region Atributi

        /// <summary>
        /// Trenutno aktivni shading model.
        /// </summary>
        private ShadeModel m_selectedModel;

        //Primitive SharpGL-a koje iscrtavamo u sceni.
        private Sphere sphere ;
        private Cylinder cyl ;

        #endregion

        #region Konstruktori

        /// <summary>
        ///          Konstruktor opengl sveta.
        /// </summary>
        public World()
        {
            sphere = new Sphere();
            cyl = new Cylinder();
            sphere.Radius = 1f;
            m_selectedModel = ShadeModel.Smooth;
        }

        #endregion

        #region Metode

        /// <summary>
        /// Korisnicka inicijalizacija i podesavanje OpenGL parametara
        /// </summary>
```

```
        public void Initialize(OpenGL gl)
        {
            gl.Enable(OpenGL.GL_DEPTH_TEST);
            gl.Enable(OpenGL.GL_CULL_FACE);

            SetupLighting(gl);

            gl.ClearColor(0f, 0f, 0f, 1.0f);

            sphere.CreateInContext(gl);
            cyl.CreateInContext(gl);

            sphere.Material = new Material();
            sphere.Material.Diffuse = Color.Red;
            sphere.Material.Ambient = Color.Blue;
            sphere.Material.Specular = Color.Green;
            sphere.Material.Shininess = 100f;

            cyl.Material = new Material();
            cyl.Material.Diffuse = Color.Red;
            cyl.Material.Ambient = Color.Green;
            cyl.Material.Specular = Color.White;
            cyl.Material.Shininess = 10f;

            gl.ShadeModel(ShadeModel.Flat);
            m_selectedModel = ShadeModel.Flat;
        }

        /// <summary>
        /// Podesavanje osvetljenja
        /// </summary>
        private void SetupLighting(OpenGL gl)
        {
            float[] global_ambient = new float[] { 0.2f, 0.2f, 0.2f,
1.0f };
            gl.LightModel(OpenGL.GL_LIGHT_MODEL_AMBIENT,
global_ambient);

            float[] light0pos = new float[] { 0.0f, 0.0f, -4.0f, 1.0f
};
            float[] light0ambient = new float[] { 0.4f, 0.4f, 0.4f,
1.0f };
            float[] light0diffuse = new float[] { 0.3f, 0.3f, 0.3f,
1.0f };
            float[] light0specular = new float[] { 0.8f, 0.8f, 0.8f,
1.0f };

            gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_POSITION, light0pos);
            gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_AMBIENT,
light0ambient);
```

```
            gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_DIFFUSE,
light0diffuse);
            gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_SPECULAR,
light0specular);
            gl.Enable(OpenGL.GL_LIGHTING);
            gl.Enable(OpenGL.GL_LIGHT0);
        }


        /// <summary>
        /// Podesava viewport i projekciju za OpenGL kontrolu.
        /// </summary>
        public void Resize(OpenGL gl, int width, int height)
        {
            gl.MatrixMode(OpenGL.GL_PROJECTION);
            gl.LoadIdentity();
            gl.Perspective(45f, (double)width / height, 0.1f, 500f);
            gl.MatrixMode(OpenGL.GL_MODELVIEW);
        }

        /// <summary>
        ///  Iscrtavanje OpenGL kontrole.
        /// </summary>
        public void Draw(OpenGL gl)
        {
            gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT |
OpenGL.GL_DEPTH_BUFFER_BIT);      // Clear The Screen And The Depth
Buffer

            gl.LoadIdentity();                                  // Reset
The View

            DrawGrid(gl);

            gl.PushMatrix();

            gl.Translate(-2f, 0f, -5f);

            sphere.Material.Bind(gl);
            /*  Deo izvornog koda SharpGL-a od klase Material
             *
https://github.com/dwmkerr/sharpgl/blob/master/source/SharpGL/Core/Sha
rpGL.SceneGraph/Assets/Material.cs
             *
             *          public void Bind(OpenGL gl)
                        {
                            //    Set the material properties.
                            gl.Material(OpenGL.GL_FRONT_AND_BACK,
OpenGL.GL_AMBIENT, ambient);
```

```
                            gl.Material(OpenGL.GL_FRONT_AND_BACK,
OpenGL.GL_DIFFUSE, diffuse);
                            gl.Material(OpenGL.GL_FRONT_AND_BACK,
OpenGL.GL_SPECULAR, specular);
                            gl.Material(OpenGL.GL_FRONT_AND_BACK,
OpenGL.GL_EMISSION, emission);
                            gl.Material(OpenGL.GL_FRONT_AND_BACK,
OpenGL.GL_SHININESS, shininess);

                            //  If we have a texture, bind it.
                            //  No need to push or pop it as we do
that earlier.
                            if (texture != null)
                                texture.Bind(gl);
                        }
              */
            sphere.Render(gl,
SharpGL.SceneGraph.Core.RenderMode.Render);

            gl.Translate(4f, -1f, 0f);
            gl.Rotate(-90f, 0f, 0f);

            cyl.Material.Bind(gl);
            cyl.Render(gl, SharpGL.SceneGraph.Core.RenderMode.Render);

            gl.PopMatrix();

            gl.Flush();
        }

        public void ChangeShadeModel(OpenGL gl)
        {
            if (m_selectedModel == ShadeModel.Flat)
            {
                gl.ShadeModel(ShadeModel.Smooth);
                m_selectedModel = ShadeModel.Smooth;
            }
            else
            {
                gl.ShadeModel(ShadeModel.Flat);
                m_selectedModel = ShadeModel.Flat;
            }
        }

}

}
```

Listing 7.2 Primer korišćenja *glMaterialfv* metode za definisanje materijala



Slika 7.2 Rezultat aplikacije iz listinga 7.2

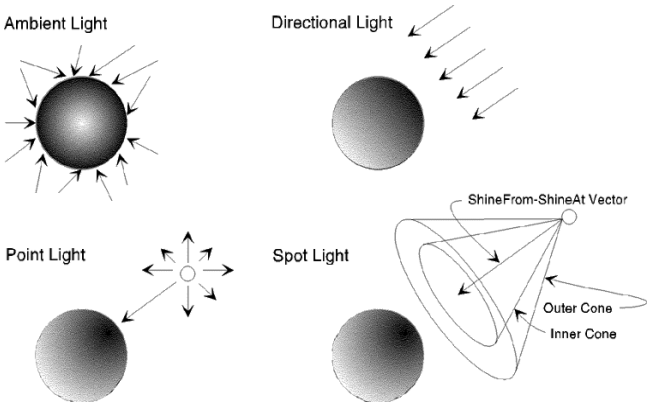
8 Osvetljenje

Osvetljenje je esencijalno u postizanju realizma modelovane scene. Takođe, svojstva materijala dolaze do izražaja tek kada je u sceni prisutno osvetljenje. Svetlosni izvori se dele u dve velike grupe: pozicioni i direkcioni. Pozicioni svetlosni izvori se dalje dele na: reflektorske i tačkaste izvore. Direkcioni svetlosni izvor podrazumeva postojanje paralelnih zraka i predstavlja svetlost koja je veoma udaljena i kod koje nema slabljenja intenziteta. Ovaj tip svetlosnog izvora simulira sunčeve zrake.

Reflektorski svetlosni izvor podrazumeva postojanje snopa zraka svetlosti. Za ovaj tip izvora je moguće definisanje dve vrste slabljenja. Prvo predstavlja slabljenje koje zavisi od udaljenosti od objekata, a drugo podrazumeva slabljenje od središta snopa ka njegovom obodu. Ovaj svetlosni izvor je moguće opisati primerom automobilskog fara.

Tačkasti svetlosni izvor karakteriše postojanje svetlosnih zraka, koji se rasejavaju na sve strane. Tačkasti svetlosni izvor predstavlja specijalni slučaj reflektorskog izvora, obzirom da je njegov cutoff ugao 180°. Slično kao i kod reflektorskog svetlosnog izvora, za njega je moguće specifikovati način slabljenja u zavisnosti od udaljenosti od objekata. Primer za njega jeste sijalica, koja nije zaklonjena preprekom i rasejava zrake na sve strane.

Slika 8.1 ilustruje tipove svetla koji se mogu kreirati u OpenGL-u.



Slika 8.1 Tipovi svetla u OpenGL-u

Pored navedenih tipova svetlosnih izvora, moguće je navesti i globalno ambijentalno osvetljenje. Ovaj tip osvetljenja služi za aproksimaciju efekta svetlosti u celoj sceni, koje se raspršuje reflektovanjem od višestrukih difuznih površina. Zraci ambijentalnog osvetljenja obuhvataju sve površine iz svih pravaca, zbog čega ne postoji senčenje.

U OpenGL standardu se proračun osvetljenja uključuje pozivom funkcije glEnable sa argumentom GL_LIGHTING. Da bi se nakon toga isključilo, potrebno je pozvati glDisable sa istim parametrom.

Za proračun osvetljenosti nekog poligona potrebno je poznavati njegovu normalu. Normala se definiše za svako teme objekta i mora biti normalizovana (jedinična normala). Moguće je definisati samo jednu normalu za poligon – normala na ravan u kojoj poligon leži (OpenGL dozvoljava samo planarne poligone!). Normala se u OpenGL standardu definiše pomoću familije funkcija glNormal, od kojih se najčešće koristi:

```
public static void glNormal3f(float nx,
                              float ny, float nz)
```

gde su:

`nx, ny, nz` – koordinate normale u normalizovanoj formi [0,1].

Određivanje vektora normale u opštem slučaju nije trivijalan zadatak. Za određivanje vektora normale na nivou poligona koristiće se pomoćna metoda koju samostalno kreiramo unutar LightingUtils klase, **FindFaceNormal**. Ova metoda računa vektor koji je ortogonalan na ravan kojem pripada poligon kreiran od tri koordinate.

```
public static float[] FindFaceNormal(
                      float x1, float y1, float z1,
                      float x2, float y2, float z2,
                      float x3, float y3, float z3)
```

gde su:

`x1,y1,z1,x2,y2,z2,x3,y3,z3` – tri temena koje određuju ravan poligona.

Transformacije sabijanja/rastezanja modela se primenjuju i na njegove vektore normala, tako da je potrebno izvršiti normalizaciju nad vektorima normala objekta nakon primene transformacija. OpenGL omogućava automatsku normalizaciju pomoću promenljive stanja *GL_NORMALIZE*. Međutim, primena automatske normalizacije smanjuje brzinu proračuna osvetljenja zbog računske složenosti. Najbolje je odmah zadavati normalizovane vektore normala.

Za definisanje modela osvetljenja scene koristi se metoda **glLightModelfv**. Uglavnom, ova metoda se koristi za zadavanje ambijentalne komponente osvetljenja u celu scenu.

```
public static void glLightModelfv(int pname,
                                  float[] params)
```

gde su:

`pname` – model osvetljenja, ima više vrednosti, najčešće se koristi *GL_LIGHT_MODEL_AMBIENT*.

`params` – semantiku određuje parametar *pname*, za gore navedenu vrednost *params* je boja ambijentalnog osvetljenja scene.

U sceni može postojati najviše jedan globalni ambijentalni svetlosni izvor.

Konkretan svetlosni izvor koji se može pozicionirati u scenu se definiše funkcijom **glLightfv**, koja ima oblik:

```
public static void glLightfv(int light, int pname,
                             float[] params)
```

gde su:

`light` - identifikator svetlosnog izvora - simboličko ime oblika:

GL_LIGHT<x>

gde je:

<x> broj ≤ *GL_MAX_LIGHTS* i <x> ≥ 0.

`pname` – model osvetljenja, moguće vrednosti su:*GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION, GL_SPOT_DIRECTION, GL_SPOT_EXPONENT, GL_SPOT_CUTOFF, GL_LINEAR_ATTENUATION* ,

GL_QUADRATIC_ATTENUATION ,
GL_CONSTANT_ATTENUATION.

`params` – semantiku određuje parametar *pname*, npr. za vrednost *params* jednako *GL_AMBIENT* je boja ambijentalne komponente svetlosnog izvora.

Nakon specifikovanja svetlosnog izvora potrebno ga je uključiti pozivom metode *glEnable* sa argumentom *GL_LIGHT<x>*, gde je <x> broj svetlosnog izvora. OpenGL specifikacija definiše minimalno osam svetlosnih izvora koje implementacija mora podržavati. Maksimum svetlosnih izvora koje implementacija podržava je sadržan u konstanti *GL_MAX_LIGHTS* koja mora biti veća ili jednaka od osam. Na Windows platformi moguće je maksimalno definisati osam svetlosnih izvora. Ako je potrebno na sceni imati više svetlosnih izvora, tada se svetlosni izvori moraju reciklirati ili se mora simulirati efekat osvetljenja.

Svetlosni izvor se pozicionira u sceni pomoću metode *glLightfv* sa *pname* jednako *GL_POSITION* i kao *params* se navodi pozicija svetlosnog izvora. Pozicija svetlosnog izvora je opisana kao uređena četvorka (*x,y,z,w*) gde su *x,y* i *z* koordinate svetlosnog izvora, a parametar *w* definiše da li je u pitanju **reflektorski** ili **direkcioni svetlosni izvor**. Vrednost nula odgovara direkcionom, a vrednost jedan reflektorskom svetlosnom izvoru. Ako se ne navede pozicija svetlosnog izvora OpenGL podrazumeva vrednost (0, 0, 1, 0) tj. direkcioni svetlosni izvor u pravcu *z*-ose.

Transformacije nad *Modelview* matricom se primenjuju i na poziciju svetlosnog izvora.

Sledeći programski kôd definiše i pozicionira direkcioni svetlosni izvor (sa identifikatorom *GL_LIGHT0*):

```
float[] pozicija = {-50.f, 50.0f, 100.0f, 0.0f};
gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_POSITION,
pozicija);
```

Ostali parametri reflektorskog svetlosnog izvora se definišu korišćenjem *GL_SPOT_DIRECTION, GL_SPOT_CUTOFF* i *GL_SPOT_EXPONENT* vrednosti za definisanje smera, *cut-off* ugla i koeficijenta opadanja intenziteta svetlosti. *Cut-off* ugao je u opsegu [0^o, 90^o] ili 180^o. Vrednost 180^o za ugao je specijalna i označava uniformno rasipanje svetlosti – **tačkasti svetlosni izvor**. Koeficijent opadanja intenziteta svetlosti, u opsegu [0, 128], definiše koliko će svetlosni snop biti fokusiran. Podrazumevane vrednosti su: za smer (0,0,-1) tj. u smeru –*z*-ose; za ugao 180^o i za koeficijent opadanja intenziteta vrednost 0.

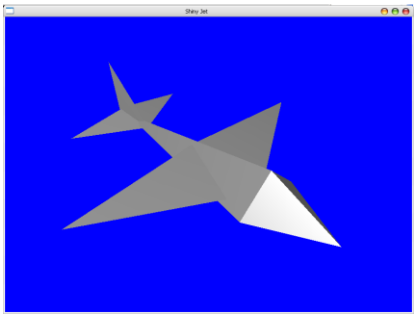
Sledeći programski kôd prikazuje postupak kreiranja reflektorskog svetlosnog izvora, njegovo pozicioniranje i uključivanje:

```
float[]  ambijentalnaKomponenta = { 0.3f, 0.3f, 0.3f, 1.0f };
float[]  difuznaKomponenta = { 0.7f, 0.7f, 0.7f, 1.0f };
float[]  smer = { 0.0f, 0.0f, -1.0f };
// Pridruži komponente svetlosnom izvoru 0
gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_AMBIENT,
             ambijentalnaKomponenta);
gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_DIFFUSE,
difuznaKomponenta);
// Podesi parametre reflektorkskog izvora
gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_SPOT_DIRECTION, smer);
gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_SPOT_CUTOFF, 45.0f);

// Ukljuci svetlosni izvor
gl.Enable(OpenGL.GL_LIGHT0);
// Pozicioniraj svetlosni izvor
float[] pozicija = { -50.f, 50.0f, 100.0f, 1.0f };
gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_POSITION, pozicija);
```

Sledeći programski kôd prikazuje postupak kreiranja tačkastog izvora svetlosti:

```
float[]  ambijentalnaKomponenta = { 0.3f, 0.3f, 0.3f, 1.0f };
float[]  difuznaKomponenta = { 0.7f, 0.7f, 0.7f, 1.0f };
// Pridruži komponente svetlosnom izvoru 0
gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_AMBIENT,
             ambijentalnaKomponenta);
gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_DIFFUSE,
difuznaKomponenta);
// Podesi parametre tackastog svetlosnog izvora
gl.Ligh(OpenGL.GL_LIGHT0, OpenGL.GL_SPOT_CUTOFF, 180.0f);
// Ukljuci svetlosni izvor
```

```
gl.Enable(OpenGL.GL_LIGHT0);
// Pozicioniraj svetlosni izvor
float[] pozicija = { -50.f, 50.0f, 100.0f, 1.0f };
```



Slika 8.5 Rezultat primera iz listinga 8.1

```
Gl.glLightfv(Gl.GL_LIGHT0, Gl.GL_POSITION, pozicija);
```

Listing 8.1. prikazuje programski kôd primera primene materijala i osvetljenja (direkciono i ambijentalno) na model aviona, preuzet iz [3]. Avion se može rotirati korišćenjem strelica na tastaturi da bi se bolje video uticaj osvetljenja. Slika 8.5 prikazuje rezultat aplikacije. Postoje dve funkcije koje iscrtavaju avion. U `DrawPlanePerFaceNormals` se bira jedinstvena normal za čitav poligon, dok se u `DrawPlanePerVertexNormals` definiše normala za svaki verteks, I ona se onda usrednjava prilikom proračuna osvetljenja.

```
using System;
using SharpGL;

namespace Lighting
{
    /// <summary>
    ///  Nabrojani tip OpenGL podrzanih tipova normala
    /// </summary>
    public enum NormalMode
    {
        PerFace,
        PerVertex
    };

    class World
    {
        #region Atributi

        /// <summary>
```

```
        ///  Ugao rotacije sveta oko X ose.
        /// </summary>
        private float m_xRotation = 0.0f;

        /// <summary>
        ///  Ugao rotacije sveta oko Y ose.
        /// </summary>
        private float m_yRotation = 0.0f;

        /// <summary>
        ///    Sirina OpenGL kontrole u pikselima.
        /// </summary>
        private int m_width;

        /// <summary>
        ///    Visina OpenGL kontrole u pikselima.
        /// </summary>
        private int m_height;

        /// <summary>
        ///    Izabrani tip normala.
        /// </summary>
        private NormalMode m_selectedNormalMode;

        /// <summary>
        ///   Konstanta koja opisuje sa koliko komponenti su zadate
pozicije temena.
        /// </summary>
        private const int VERTEX_COMPONENT_COUNT = 3;

        /// <summary>
        ///   Niz sa koordinatama temena aviona
        /// </summary>
        private float[] m_vertices = {
                                        0.0f, 0.0f, 60.0f, -15.0f, 0.0f,
30.0f, 15.0f, 0.0f, 30.0f,
                                        15.0f, 0.0f, 30.0f, 0.0f, 15.0f,
30.0f, 0.0f, 0.0f, 60.0f,
                                        0.0f, 0.0f, 60.0f, 0.0f, 15.0f,
30.0f, -15.0f, 0.0f, 30.0f,
                                        -15.0f, 0.0f, 30.0f, 0.0f,
15.0f, 30.0f, 0.0f, 0.0f, -56.0f,
                                        0.0f, 0.0f, -56.0f, 0.0f, 15.0f,
30.0f, 15.0f, 0.0f, 30.0f,
                                        15.0f, 0.0f, 30.0f, -15.0f,
0.0f, 30.0f, 0.0f, 0.0f, -56.0f,
                                        0.0f, 2.0f, 27.0f, -60.0f, 2.0f,
-8.0f, 60.0f, 2.0f, -8.0f,
                                        60.0f, 2.0f, -8.0f, 0.0f, 7.0f,
-8.0f, 0.0f, 2.0f, 27.0f,
```

```
                                        60.0f, 2.0f, -8.0f, -60.0f,
2.0f, -8.0f, 0.0f, 7.0f, -8.0f,
                                        0.0f, 2.0f, 27.0f, 0.0f, 7.0f, -
8.0f, -60.0f, 2.0f, -8.0f,
                                        -30.0f, -0.5f, -57.0f, 30.0f, -
0.5f, -57.0f, 0.0f, -0.5f, -40.0f,
                                        0.0f, -0.5f, -40.0f, 30.0f, -
0.5f, -57.0f, 0.0f, 4.0f, -57.0f,
                                        0.0f, 4.0f, -57.0f, -30.0f, -
0.5f, -57.0f, 0.0f, -0.5f, -40.0f,
                                        30.0f, -0.5f, -57.0f, -30.0f, -
0.5f, -57.0f, 0.0f, 4.0f, -57.0f,
                                        0.0f, 0.5f, -40.0f, 3.0f, 0.5f,
-57.0f, 0.0f, 25.0f, -65.0f,
                                        0.0f, 25.0f, -65.0f, -3.0f,
0.5f, -57.0f, 0.0f, 0.5f, -40.0f,
                                        3.0f, 0.5f, -57.0f, -3.0f, 0.5f,
-57.0f, 0.0f, 25.0f, -65.0f
                                    };

        /// <summary>
        ///   Niz sa normalama u temenima aviona
        /// </summary>
        private float[] m_normals;

        #endregion Atributi

        #region Properties

        /// <summary>
        ///    Izabrani tip normala.
        /// </summary>
        public NormalMode SelectedNormalMode
        {
            get { return m_selectedNormalMode; }
            set { m_selectedNormalMode = value; }
        }

        /// <summary>
        ///    Ugao rotacije sveta oko Y ose.
        /// </summary>
        public float RotationY
        {
            get { return m_yRotation; }
            set { m_yRotation = value; }
        }

        /// <summary>
        ///    Ugao rotacije sveta oko X ose.
        /// </summary>
```

```
        public float RotationX
        {
            get { return m_xRotation; }
            set { m_xRotation = value; }
        }

        /// <summary>
        ///    Sirina OpenGL kontrole u pikselima.
        /// </summary>
        public int Width
        {
            get { return m_width; }
            set { m_width = value; }
        }

        /// <summary>
        ///    Visina OpenGL kontrole u pikselima.
        /// </summary>
        public int Height
        {
            get { return m_height; }
            set { m_height = value; }
        }

        #endregion Properties

        #region Metode

        /// <summary>
        /// Korisnicka inicijalizacija i podesavanje OpenGL parametara
        /// </summary>
        public void Initialize(OpenGL gl)
        {
            gl.Enable(OpenGL.GL_DEPTH_TEST);
            gl.Enable(OpenGL.GL_CULL_FACE);
            SetupLighting(gl);
            m_selectedNormalMode = NormalMode.PerFace;
        }

        /// <summary>
        /// Podesava viewport i projekciju za OpenGL kontrolu.
        /// </summary>
        public void Resize(OpenGL gl, int width, int height)
        {
            gl.MatrixMode(OpenGL.GL_PROJECTION);
            gl.LoadIdentity();
            gl.Perspective(45f, (double)width / height, 0.1f, 500f);
            gl.MatrixMode(OpenGL.GL_MODELVIEW);
        }
```

```
        /// <summary>
        /// Podesavanje osvetljenja
        /// </summary>
        private void SetupLighting(OpenGL gl)
        {
            float[] global_ambient = new float[] { 0.2f, 0.2f, 0.2f,
1.0f };
            gl.LightModel(OpenGL.GL_LIGHT_MODEL_AMBIENT,
global_ambient);

            float[] light0pos = new float[] { 0.0f, 10.0f, -10.0f,
1.0f };
            float[] light0ambient = new float[] { 0.4f, 0.4f, 0.4f,
1.0f };
            float[] light0diffuse = new float[] { 0.3f, 0.3f, 0.3f,
1.0f };
            float[] light0specular = new float[] { 0.8f, 0.8f, 0.8f,
1.0f };

            gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_POSITION, light0pos);
            gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_AMBIENT,
light0ambient);
            gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_DIFFUSE,
light0diffuse);
            gl.Light(OpenGL.GL_LIGHT0, OpenGL.GL_SPECULAR,
light0specular);
            gl.Enable(OpenGL.GL_LIGHTING);
            gl.Enable(OpenGL.GL_LIGHT0);

            // Definisemo belu spekularnu komponentu materijala sa
jakim odsjajem
            gl.Material(OpenGL.GL_FRONT, OpenGL.GL_SPECULAR,
light0specular);
            gl.Material(OpenGL.GL_FRONT, OpenGL.GL_SHININESS, 128.0f);

            //Uikljuci color tracking mehanizam
            gl.Enable(OpenGL.GL_COLOR_MATERIAL);

            // Podesi na koje parametre materijala se odnose pozivi
glColor funkcije
            gl.ColorMaterial(OpenGL.GL_FRONT,
OpenGL.GL_AMBIENT_AND_DIFFUSE);

            // Ukljuci automatsku normalizaciju nad normalama
            gl.Enable(OpenGL.GL_NORMALIZE);

            m_normals =
LightingUtilities.ComputeVertexNormals(m_vertices);

            gl.ShadeModel(OpenGL.GL_SMOOTH);
```

```
        }
        /// <summary>
        ///  Iscrtavanje OpenGL kontrole.
        /// </summary>
        public void Draw(OpenGL gl)
        {
            gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT |
OpenGL.GL_DEPTH_BUFFER_BIT);

            gl.LoadIdentity();

            gl.Translate(0.0f, 0.0f, -150.0f);
            gl.Rotate(m_yRotation, 0f, 1f, 0f);
            gl.Rotate(m_xRotation, 1f, 0f, 0f);

            //Podesavanje boje za anbient i diffuse komponentu
osvetljenja
            gl.Color(0.4f, 0.4f, 0.4f);

            switch (m_selectedNormalMode)
            {
                case (NormalMode.PerFace):
                    {
                        DrawPlanePerFaceNormals(gl);
                        break;
                    }
                case (NormalMode.PerVertex):
                    {
                        DrawPlanePerVertexNormals(gl);
                        break;
                    }
            }
        }


        private void DrawPlanePerVertexNormals(OpenGL gl)
        {
            // Ukljucivanje rada sa vertex array (VA) mehanizmom
            gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);

            //Ukljucivanje rada sa normal array mehanizmom.
            //Niz sa normalama se zadaje na isti nacin kao i niz sa
koordinatama temena.
            gl.EnableClientState(OpenGL.GL_NORMAL_ARRAY);

            // namesti pointer na nizove temena i normala
            gl.VertexPointer(VERTEX_COMPONENT_COUNT, 0, m_vertices);
            gl.NormalPointer(OpenGL.GL_FLOAT, 0, m_normals);
```

```
            gl.DrawArrays(OpenGL.GL_TRIANGLES, 0, m_vertices.Length /
VERTEX_COMPONENT_COUNT);

            //  Iskljucivanje postojeceg pokazivaca na niz sa
pozicijama temena i niz sa normalama
            gl.VertexPointer(VERTEX_COMPONENT_COUNT, OpenGL.GL_FLOAT,
0, IntPtr.Zero);
            gl.NormalPointer(OpenGL.GL_FLOAT, 0, IntPtr.Zero);

            //Iskljucivanje rada sa vertex array i normal array
mehanizmom
            gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
            gl.DisableClientState(OpenGL.GL_NORMAL_ARRAY);
        }


        /// <summary>
        ///  Iscrtavanje aviona.
        /// </summary>
        private void DrawPlanePerFaceNormals(OpenGL gl)
        {
            // Nose Cone - gleda na dole
            gl.Begin(OpenGL.GL_TRIANGLES);

            //nekad je moguce direktno odrediti normalu
            gl.Normal(0.0f, -1.0f, 0.0f);
            gl.Vertex(0.0f, 0.0f, 60.0f);
            gl.Vertex(-15.0f, 0.0f, 30.0f);
            gl.Vertex(15.0f, 0.0f, 30.0f);
            gl.End();

            gl.Begin(OpenGL.GL_TRIANGLES);
            gl.Normal(LightingUtilities.FindFaceNormal(15.0f, 0.0f,
30.0f, 0.0f, 15.0f, 30.0f, 0.0f, 0.0f, 60.0f));
            gl.Vertex(15.0f, 0.0f, 30.0f);
            gl.Vertex(0.0f, 15.0f, 30.0f);
            gl.Vertex(0.0f, 0.0f, 60.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(0.0f, 0.0f,
60.0f, 0.0f, 15.0f, 30.0f, -15.0f, 0.0f, 30.0f));
            gl.Vertex(0.0f, 0.0f, 60.0f);
            gl.Vertex(0.0f, 15.0f, 30.0f);
            gl.Vertex(-15.0f, 0.0f, 30.0f);

            // Body of the Plane
            gl.Normal(LightingUtilities.FindFaceNormal(-15.0f, 0.0f,
30.0f, 0.0f, 15.0f, 30.0f, 0.0f, 0.0f, -56.0f));
            gl.Vertex(-15.0f, 0.0f, 30.0f);
            gl.Vertex(0.0f, 15.0f, 30.0f);
            gl.Vertex(0.0f, 0.0f, -56.0f);
```

```
            gl.Normal(LightingUtilities.FindFaceNormal(0.0f, 0.0f, -
56.0f, 0.0f, 15.0f, 30.0f, 15.0f, 0.0f, 30.0f));
            gl.Vertex(0.0f, 0.0f, -56.0f);
            gl.Vertex(0.0f, 15.0f, 30.0f);
            gl.Vertex(15.0f, 0.0f, 30.0f);

            gl.Normal(0.0f, -1.0f, 0.0f);
            gl.Vertex(15.0f, 0.0f, 30.0f);
            gl.Vertex(-15.0f, 0.0f, 30.0f);
            gl.Vertex(0.0f, 0.0f, -56.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(0.0f, 2.0f,
27.0f, -60.0f, 2.0f, -8.0f, 60.0f, 2.0f, -8.0f));
            gl.Vertex(0.0f, 2.0f, 27.0f);
            gl.Vertex(-60.0f, 2.0f, -8.0f);
            gl.Vertex(60.0f, 2.0f, -8.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(60.0f, 2.0f, -
8.0f, 0.0f, 7.0f, -8.0f, 0.0f, 2.0f, 27.0f));
            gl.Vertex(60.0f, 2.0f, -8.0f);
            gl.Vertex(0.0f, 7.0f, -8.0f);
            gl.Vertex(0.0f, 2.0f, 27.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(60.0f, 2.0f, -
8.0f, -60.0f, 2.0f, -8.0f, 0.0f, 7.0f, -8.0f));
            gl.Vertex(60.0f, 2.0f, -8.0f);
            gl.Vertex(-60.0f, 2.0f, -8.0f);
            gl.Vertex(0.0f, 7.0f, -8.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(0.0f, 2.0f,
27.0f, 0.0f, 7.0f, -8.0f, -60.0f, 2.0f, -8.0f));
            gl.Vertex(0.0f, 2.0f, 27.0f);
            gl.Vertex(0.0f, 7.0f, -8.0f);
            gl.Vertex(-60.0f, 2.0f, -8.0f);

            gl.Normal(0.0f, -1.0f, 0.0f);
            gl.Vertex(-30.0f, -0.50f, -57.0f);
            gl.Vertex(30.0f, -0.50f, -57.0f);
            gl.Vertex(0.0f, -0.50f, -40.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(0.0f, -0.5f, -
40.0f, 30.0f, -0.5f, -57.0f, 0.0f, 4.0f, -57.0f));
            gl.Vertex(0.0f, -0.5f, -40.0f);
            gl.Vertex(30.0f, -0.5f, -57.0f);
            gl.Vertex(0.0f, 4.0f, -57.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(0.0f, 4.0f, -
57.0f, -30.0f, -0.5f, -57.0f, 0.0f, -0.5f, -40.0f));
            gl.Vertex(0.0f, 4.0f, -57.0f);
```

```
            gl.Vertex(-30.0f, -0.5f, -57.0f);
            gl.Vertex(0.0f, -0.5f, -40.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(30.0f, -0.5f, -
57.0f, -30.0f, -0.5f, -57.0f, 0.0f, 4.0f, -57.0f));
            gl.Vertex(30.0f, -0.5f, -57.0f);
            gl.Vertex(-30.0f, -0.5f, -57.0f);
            gl.Vertex(0.0f, 4.0f, -57.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(0.0f, 0.5f, -
40.0f, 3.0f, 0.5f, -57.0f, 0.0f, 25.0f, -65.0f));
            gl.Vertex(0.0f, 0.5f, -40.0f);
            gl.Vertex(3.0f, 0.5f, -57.0f);
            gl.Vertex(0.0f, 25.0f, -65.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(0.0f, 25.0f, -
65.0f, -3.0f, 0.5f, -57.0f, 0.0f, 0.5f, -40.0f));
            gl.Vertex(0.0f, 25.0f, -65.0f);
            gl.Vertex(-3.0f, 0.5f, -57.0f);
            gl.Vertex(0.0f, 0.5f, -40.0f);

            gl.Normal(LightingUtilities.FindFaceNormal(3.0f, 0.5f, -
57.0f, -3.0f, 0.5f, -57.0f, 0.0f, 25.0f, -65.0f));
            gl.Vertex(3.0f, 0.5f, -57.0f);
            gl.Vertex(-3.0f, 0.5f, -57.0f);
            gl.Vertex(0.0f, 25.0f, -65.0f);
            gl.End();
        }

        public void SwitchNormalMode()
        {
            if (m_selectedNormalMode == NormalMode.PerFace)
            {
                m_selectedNormalMode = NormalMode.PerVertex;
            }
            else
            {
                m_selectedNormalMode = NormalMode.PerFace;
            }
        }

        #endregion
    }
}
```

Listing 8.1 Iscrtavanje aviona sa proračunom normala

U stvarnosti objekti koji su osvetljeni „bacaju" senku. OpenGL model osvetljenja ne podržava automatsko generisanje senki, tako da autor mora sam generisati i prikazati senku. Proračun i prikaz senki je van okvira ovog teksta. O senkama čitaoci se mogu više informisati u [3].

OpenGL podržava definisanje efekta magle. Proračun efekta magle se uključuje pozivom metode *glEnable* sa argumentom *GL_FOG*. Familija metoda **glFog** određuje parametre magle.

```
public static void glFogi(int pname, int param)
public static void glFogf(int pname, float param)
public static void glFogfv(int pname,float[] param)
```

gde su:

`pname` – parametar koji definiše koja se osobina magle podešava. Neke od mogućih vrednosti su:

- *GL_FOG_MODE* – *params* je *blending* faktor (jednačina po kojoj se računa magla) koji može biti: *GL_LINEAR, GL_EXP*, and *GL_EXP2*.
- *GL_FOG_DENSITY* – *params* je gustina magle, vrednost mora biti nenegativna; inicijalno iznosi 1.
- *GL_FOG_START* – *params* je distanca od koje se počinje primenjivati magla; inicijalno iznosi 0.
- *GL_FOG_END* – *params* je distanca nakon koje svi objekti su u magli; inicijalno iznosi 1.
- *GL_FOG_COLOR* – *params* definiše boju magle; inicijalno je (0, 0, 0, 0) – crna boja.

`param` – semantiku određuje parametar *pname*.

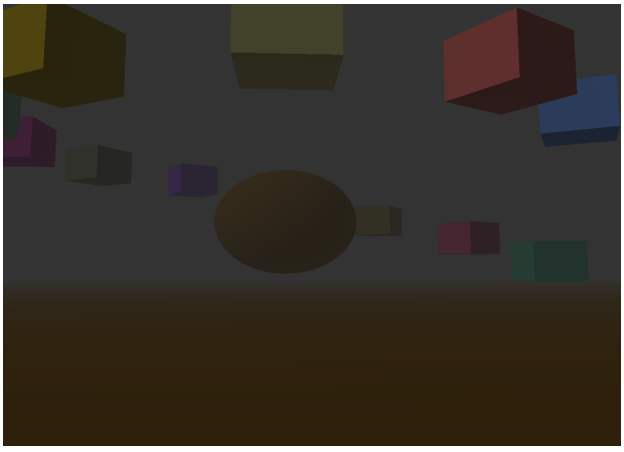
Listing 8.3 prikazuje isečak programskog kôda primera primene efekta magle. Slika 8.7 prikazuje rezultat ovog primera. Primer predstavlja proširenje primera upravljanja kamerom.

```
        /// <summary>
        /// Podesavanje efekta magle
        /// </summary>
        public void SetupFog(OpenGL gl)
        {
            float[] grayLight = { 0.2f, 0.2f, 0.2f, 1.0f }; // Boja
magle = boja pozadine
```

```
            //Kreiranje efekta magle
            gl.Enable(OpenGL.GL_FOG);                   // ukljuci
efekat magle
            gl.Fog(OpenGL.GL_FOG_COLOR, grayLight);    // boja magle =
boja pozadine
            gl.Fog(OpenGL.GL_FOG_START, 1.0f);         // magla pocinje
od 5 jedinica
            gl.Fog(OpenGL.GL_FOG_END, 20.0f);          // magla
prestaje nakon 50 jedinica
            gl.Fog(OpenGL.GL_FOG_MODE, OpenGL.GL_LINEAR); // jednacina
za proracun efekta magle
        }
```

Listing 8.3 Primer kreiranja efekta magle

OpenGL omogućava simuliranje različitih svetlosnih efekata korišćenjem **blending** tehnike. Više o ovoj tehnici se može naći u [3]. Dodatno, OpenGL podržava i *dithering*, primenu logičkih operacija prilikom kombinovanja boja, kao i *color masking*. Više detalja se može naći u [3].



Slika 8.7 Rezultat primera iz listinga 8.3

9 Teksture

Pridruživanje teksture objektima doprinosi realizmu scene. U ovom poglavlju biće prezentovane metode za učitavanje tekstura i njihovo mapiranje. Jedinica teksture se naziva **teksel**.

OpenGL podržava rad sa 1D, 2D i 3D teksturama, koje se učitavaju pomoću metoda **glTexImage1D**, **glTexImage2D** i **glTexImage3D**.

```
public static void glTexImage1D(int target,
                 int level, int internalformat,
                int width, int border,
                int format, int type, IntPtr data)

public static void glTexImage2D(int target,
                  int level, int internalformat,
                 int width, int height,
                 int border, int format,
                 int type, IntPtr data)

public static void glTexImage3D(int target,
                int level, int internalformat,
                 int width, int height,
                 int depth, int border,
                 int format, int type,
                 IntPtr data)
```

gde su:

`target` – odgovarajuća vrednost shodno dimenzionalnosti teksture: *GL_TEXTURE_1D*, *GL_TEXTURE_2D* i *GL_TEXTURE_3D*,

`level` – *mipmaping level* (ako se ne koristi stavlja se 0),

`internalFormat` – format teksela,

`width, height i depth` – dimenzije teksture,

`border` – širina okvira. Dozvoljene vrednosti su: 0 i 1,

`format, type i data` – format, tip piksela i pikseli pikselmape koja se koristi kao tekstura.

Teksture se mogu kreirati na osnovu sadržaja kolor bafera (iscrtanog dela kadra) pomoću **glCopyTexImage1D** i **glCopyTexImage2D** metoda.

```
public static void glCopyTexImage1D(int target,
            int level, int internalFormat,
            int x, int y, int width, int border)
public static void glCopyTexImage2D(int target,
                  int level, int internalFormat,
                  int x, int y,
                   int width, int height,
                   int border)
```

gde su:

`target` - odgovarajuća vrednost shodno dimenzionalnosti teksture: *`GL_TEXTURE_1D`* i *`GL_TEXTURE_2D`*,
`level` – *mipmaping level* (ako se ne koristi stavlja se 0),
`internalFormat` – format teksela,
`x i y` – koordinate donje-levog temena regiona koji se kopira iz kolor bafera,
`width, height` – dimenzije teksture,
`border` – širina okvira.Dozvoljene vrednosti su: 0 i 1.

Često se teksture menjaju u toku izvršavanja, učitavaju nove i uklanjaju stare iz memorije. Gore navedene operacije za učitavanje teksture su zahtevne po pitanju resursa. S toga postoje metode **glTexSubImage1D**, **glTexSubImage2D** i **glTexSubImage3D** koje omogućavaju izdvajanje delova tekstura koje su već učitane u memoriju.

```
public static void glTexSubImage1D(int target,
            int level, int xOffset, int width,
            int format, int type, IntPtr data)
public static void glTexSubImage2D(int target,
            int level, int xOffset, int yOffset,
            int width, int height, int format,
            int type, IntPtr data)
public static void glTexSubImage3D(int target,
      int level, int xOffset, int yOffset,
```

```
int zOffset, int width, int height,
int depth, int format, int type, IntPtr data)
```

gde su:

`target` – odgovarajuća vrednost shodno dimenzionalnosti teksture: *GL_TEXTURE_1D, GL_TEXTURE_2D i GL_TEXTURE_3D*,

`level` – *mipmaping level* (ako se ne koristi stavlja se 0),

`xOffset, zOffset i yOffset` – pozicija u staroj teksturi od koje se vrši zamena,

`width, height i depth` – dimenzije teksture,

`format, type i data` – su format, tip piksela i pikseli pikselmape koja se koristi kao tekstura.

Moguće je zameniti teksturu delom iscrtanog kadra u kolor baferu sa metodama **glCopyTexSubImage1D** i **glCopyTexSubImage2D**.

Za mapiranje teksture na objekat (tj. teksela na temena objekta) koriste se koordinate teksture. Slično X3D standardu, koordinate teksture se izražavaju u (*s*,*t*,*r*) komponentama koje uzimaju vrednost iz opsega [0,1]. 1D teksture imaju samo *s* komponentu, 2D komponente *s* i *t*, a 3D teksture imaju *s,t* i *r* komponente. U režim mapiranja dvodimenzionalne teksture se ulazi pozivom metode *glEnable* sa argumentom *GL_TEXTURE_2D*. Analogno se ulazi u režime mapiranja za teksture drugih dimenzionalnosti.

Koordinate 1D, 2D i 3D teksture se specifikuju pomoću metoda **glTexCoord1f**, **glTexCoord2f** i **glTexCoord3f**.

```
public static void glTexCoord1f(float s)

public static void glTexCoord2f(float s, float t)

public static void glTexCoord3f(float s, float t,
                                 float r)
```

gde su:

`s,t,r` – koordinate teksture (širina, visina, dubina).

Za objekte relativno složene geometrije ručno zadavanje koordinata teksture je veoma teško i podložno greškama. OpenGL standard nudi mogućnost automatskog generisanja koordinata teksture. Automatsko generisanje koordinata tekstura se uključuje preko promenljivih stanja: GL_TEXTURE_GEN_S, GL_TEXTURE_GEN_T i GL_TEXTURE_GEN_R. Za dvodimenzionalne teksture se koriste vrednosti: GL_TEXTURE_GEN_S i GL_TEXTURE_GEN_T. Po uključivanju sve pozivi *glTexCoord** metoda se ignorišu i OpenGL automatski generiše koordinate teksture.

OpenGL implementaciji se mora zadati na koji način da generiše koordinate teksture. Ovo se postiže pozivom familije metoda **glTexGen***.

```
public static void glTexGenf(int coord, int pname,
                             float param)
public static void glTexGenfv(int coord, int pname,
                             float[] param)
```

gde su:

- `coord` – specifikuje koju koordinatu zadajemo. Dozvoljene vrednosti su: *GL_S*, *GL_T*, i *GL_R*,
- `pname` – vrednost određuje značenje *param* parametra. Moguće vrednosti su: *GL_OBJECT_PLANE* i *GL_EYE_PLANE*, i *GL_TEXTURE_GEN_MODE,*
- `param` – u slučaju kad je *pname* GL_TEXTURE_GEN_MODE, tada ovaj parametar definiše način generisanja koordinata teksture. Dozvoljene vrednosti su: *GL_OBJECT_LINEAR, GL_EYE_LINEAR, GL_SPHERE_MAP, GL_NORMAL_MAP, GL_REFLECTION_MAP*.

OpenGL podržava tri načina automatskog generisanja koordinata tekstura: linearno koje je izraženo u koordinatama objekta (*GL_OBJECT_LINEAR*), linearno koje je izraženo u koordinatama sa aspekta posmatrača (*GL_EYE_LINEAR*), i sferično (*GL_SPHERE_MAP*). Prva dva koriste ravan, a poslednja sferu. Sferično generisanje koordinata teksture daje realistične rezultate iako u praksi se češće koristi mapiranje na kocku. Više detalja o ovom mapiranju se može naći u [3].

Transformacije nad koordinatama teksture se realizuju korišćenjem **Texture** matrice. Ova matrica se bira pozivom metode *glMatrixMode* sa argumentom *GL_TEXTURE*.

OpenGL omogućava definisanje načina na koji se tekseli stapaju (*texel blending*) sa materijalom. U tu svrhu se koristi **glTexEnv** familija metoda.

```
public static void glTexEnvi(int target, int pname,
                             int param)

public static void glTexEnvf(int target, int pname,
                             float param)

public static void glTexEnviv(int target,int pname,
                             int[] param)
```

```
public static void glTexEnvfv(int target,int pname,
                               float[] param)
```

gde su:

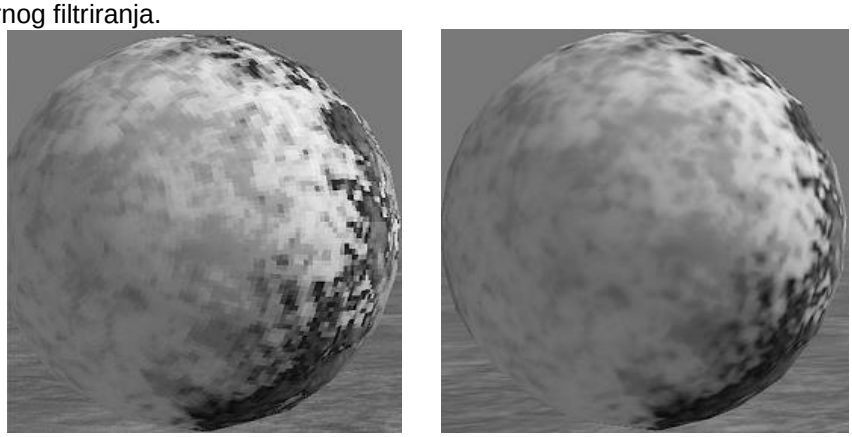
`target` – ukoliko se podešava blending, parametar mora biti *GL_TEXTURE_ENV*.

`pname` – ukoliko se podešava blending, vrednost je *GL_TEXTURE_ENV_MODE*, i tada *params* može biti:

- *GL_MODULATE* – množi teksel sa bojom materijala,
- *GL_ADD* – sabira teksel sa bojom materijala,
- *GL_DECAL* – *blending* na osnovu *alpha* komponente,
- *GL_BLEND* – *blending* teksela sa bojom materijala,
- *GL_REPLACE* – teksel zamenjuje boju materijala.

`params` – semantiku određuje parametar *pname*.

Mapiranje teksela na objekat često podrazumeva skaliranje teksela. OpenGL omogućava specifikovanje filtara za skaliranje tekstura. Filtri se definišu za smanjenje (*minimization*) i uvećavanje (*magnification*) teksture. Neki od filtera su: najbliži sused (*nearest neighbour)*, slika 9.1a, i linearno filtriranje (*linear filtering)*, slika 9.1b*.* Najbliži sused određuje vrednost nedostajućeg teksela na osnovu njegovog najbližeg suseda, dok linearno filtriranje određuje vrednost nedostajućeg teksela na osnovu srednje vrednosti okolnih teksela. Najbliži sused daje vizeulno lošiji rezultat u poređenju sa linearnim filtriranjem, ali zato je značajno brži od linearnog filtriranja.



a) najbliži sused b) linearno filtriranje

Slika 9.1 Tipovi filtriranja teksture, preuzeto iz [3]

OpenGL zahteva da se definiše kako se rukuje koordinatama teksture koje su van opsega ([0,1]) (*texture wrapping*). Jedna od mogućnosti je ponavljanje teksture u pravcu u kome su koordinate teksture izvan opsega.

Familija metoda **glTexParameter** omogućava gore navedenu funkcionalnost.

```
public static void glTexParameterf(int target,
                  int pname, float param)
public static void glTexParameteri(int target,
                              int pname, int param)

public static void glTexParameterfv(int target,
                  int pname, float[] params)
public static void glTexParameteriv(int target,
                   int pname, int[] params)
```

gde su:

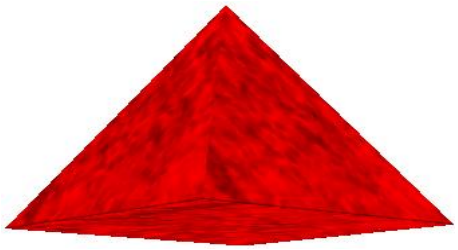
`target` – parametar mora biti:*GL_TEXTURE_1D, GL_TEXTURE_2D* ili *GL_TEXTURE_3D*.

`pname` – može biti:

- *GL_TEXTURE_MIN_FILTER* – filtriranje teksture koje se primenjuje u slučaju da se tekstura mora smanjiti da bi bila pridružena objektu,
- *GL_TEXTURE_MAG_FILTER* – filtriranje teksture koje se primenjuje u slučaju da se tekstura mora povećati da bi bila pridružena objektu,
- *GL_TEXTURE_WRAP_S* – definiše da li će se tekstura ponavljati po *s*-osi, kao i kojim tekselima će biti realizovano ponavljanje,
- *GL_TEXTURE_WRAP_T* – definiše da li će se tekstura ponavljati po *t*-osi, kao i kojim tekselima će biti realizovano ponavljanje,
- *GL_TEXTURE_BORDER_COLOR* – definiše boju okvira teksture. Podrazumevana vrednost je crna boja,
- *GL_TEXTURE_PRIORITY* – definiše prioritet teksture.

`params` – semantiku određuje parametar *pname*.

Listing 9.1 i slika 9.2 prikazuju primer pridruživanja teksture objektu – piramidi. Primer je preuzet iz [3].



Slika 9.2 Rezultat primera iz listinga 9.1

```
using System;
using SharpGL;
using System.Drawing;
using System.Drawing.Imaging;
using SharpGL.SceneGraph.Quadrics;

namespace Texturing
{

    /// <summary>
    ///  Nabrojani tip OpenGL rezima stapanja teksture sa materijalom
    /// </summary>
    public enum TextureBlendingMode
    {
        Modulate,
        Decal,
        Replace,
        Blend
    };

    /// <summary>
    ///  Klasa enkapsulira OpenGL kod i omogucava njegovo iscrtavanje
i azuriranje.
    /// </summary>
    public class World
    {
        #region Atributi

        /// <summary>
        ///  Izabrana OpenGL mehanizam za iscrtavanje.
        /// </summary>
        private TextureBlendingMode m_selectedMode =
TextureBlendingMode.Replace;

        /// <summary>
        ///  Ugao rotacije sveta oko X ose.
        /// </summary>
        private float m_xRotation = 0.0f;

        /// <summary>
        ///  Ugao rotacije sveta oko Y ose.
        /// </summary>
        private float m_yRotation = 0.0f;

        /// <summary>
```

```
        ///    Sirina OpenGL kontrole u pikselima.
        /// </summary>
        private int m_width;

        /// <summary>
        ///    Visina OpenGL kontrole u pikselima.
        /// </summary>
        private int m_height;

        #endregion Atributi

    #region Properties

        /// <summary>
        ///  Izabrani OpenGL rezim stapanja teksture sa materijalom
        /// </summary>
        public TextureBlendingMode SelectedMode
        {
            get { return m_selectedMode; }
            set
            {
                m_selectedMode = value;
            }
        }

        /// <summary>
        ///    Ugao rotacije sveta oko Y ose.
        /// </summary>
        public float RotationY
        {
            get { return m_yRotation; }
            set { m_yRotation = value; }
        }

        /// <summary>
        ///    Ugao rotacije sveta oko X ose.
        /// </summary>
        public float RotationX
        {
            get { return m_xRotation; }
            set { m_xRotation = value; }
        }

        /// <summary>
        ///    Sirina OpenGL kontrole u pikselima.
        /// </summary>
        public int Width
        {
            get { return m_width; }
            set { m_width = value; }
```

```
        }

        /// <summary>
        ///    Visina OpenGL kontrole u pikselima.
        /// </summary>
        public int Height
        {
            get { return m_height; }
            set { m_height = value; }
        }

        #endregion Properties

        #region Konstruktori

        /// <summary>
        ///  Konstruktor.
        /// </summary>
        public World()
        {
        }

        #endregion Konstruktori

        #region Metode

        private Sphere sfera;
        /// <summary>
        ///  Korisnicka inicijalizacija i podesavanje OpenGL
parametara.
        /// </summary>
        public void Initialize(OpenGL gl)
        {
            sfera = new Sphere();
            sfera.CreateInContext(gl);
            sfera.Radius = 1f;
            gl.Enable(OpenGL.GL_DEPTH_TEST);
            gl.FrontFace(OpenGL.GL_CCW);
            gl.Enable(OpenGL.GL_CULL_FACE);

            //Bela pozadina
            gl.ClearColor(1.0f, 1.0f, 1.0f, 1.0f);

            // Ucitaj sliku
            // Prozirna slika 25%
            //Bitmap image = new Bitmap("..//..//images//stone.png");

            // Neprozirna slika
            Bitmap image = new Bitmap("..//..//images//stone.jpg");
```

```
            // rotiramo sliku zbog koordinantog sistema opengl-a
            image.RotateFlip(RotateFlipType.RotateNoneFlipY);
            Rectangle rect = new Rectangle(0, 0, image.Width,
image.Height);

            // ako je potreban RGB format
             BitmapData imageData = image.LockBits(rect,
System.Drawing.Imaging.ImageLockMode.ReadOnly,

System.Drawing.Imaging.PixelFormat.Format24bppRgb);

            // Kreiraj teksturu sa RBG formatom
             gl.TexImage2D(OpenGL.GL_TEXTURE_2D, 0, OpenGL.GL_RGB8,
imageData.Width, imageData.Height, 0,
                           OpenGL.GL_BGR, OpenGL.GL_UNSIGNED_BYTE,
imageData.Scan0);


            // RGBA format (dozvoljena providnost slike tj. alfa
kanal)
       /*     BitmapData imageData = image.LockBits(rect,
System.Drawing.Imaging.ImageLockMode.ReadOnly,

System.Drawing.Imaging.PixelFormat.Format32bppArgb);

            // Kreiraj teksturu sa RGBA
            gl.TexImage2D(OpenGL.GL_TEXTURE_2D, 0,
(int)OpenGL.GL_RGBA8, imageData.Width, imageData.Height, 0,
                           OpenGL.GL_BGRA, OpenGL.GL_UNSIGNED_BYTE,
imageData.Scan0);*/

            // Podesi parametre teksture
            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_LINEAR);
            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MAG_FILTER, OpenGL.GL_LINEAR);
            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_WRAP_S, OpenGL.GL_CLAMP);
            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_WRAP_T, OpenGL.GL_CLAMP);

            // Podesi nacin blending teksture
            m_selectedMode = TextureBlendingMode.Modulate;
            gl.TexEnv(OpenGL.GL_TEXTURE_ENV,
OpenGL.GL_TEXTURE_ENV_MODE, OpenGL.GL_MODULATE);

            // Predji u rezim rada sa 2D teksturama
            gl.Enable(OpenGL.GL_TEXTURE_2D);

            // Posto je kreirana tekstura slika nam vise ne treba
```

```
            image.UnlockBits(imageData);
            image.Dispose();
        }

        /// <summary>
        /// Podesava viewport i projekciju za OpenGL kontrolu.
        /// </summary>
        public void Resize(OpenGL gl, int height, int width)
        {
            m_height = height;
            m_width = width;
            gl.Viewport(0, 0, m_width, m_height);
            gl.MatrixMode(OpenGL.GL_PROJECTION);
            gl.LoadIdentity();
            gl.Perspective(45.0, m_width / m_height, 0.1, 40.0);
            gl.MatrixMode(OpenGL.GL_MODELVIEW);
            gl.LoadIdentity();
        }
        /// <summary>
        ///  Iscrtavanje OpenGL kontrole.
        /// </summary>
        public void Draw(OpenGL gl)
        {
            gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT |
OpenGL.GL_DEPTH_BUFFER_BIT);

            gl.PushMatrix();
                gl.Translate(0f, -0.5f, 0f);
                gl.Rotate(m_xRotation, 1.0f, 0.0f, 0.0f);
                gl.Rotate(m_yRotation, 0.0f, 1.0f, 0.0f);

                gl.Scale(.5f, .5f, .5f);
                //Nacrtaj piramidu
                //uvek je dobro pravilo da se definisu materijali s
obzirom da postoji
                //mogucnost da se ne kreira i pridruzi tekstura
objektu
                gl.Color(1f, 0f, 0f, 0.2f);
                gl.Begin(OpenGL.GL_TRIANGLES);
                //Osnova piramide
                gl.TexCoord(1.0f, 1.0f);
                gl.Vertex(0.5f, 0.0f, -0.5f);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(-0.5f, 0.0f, 0.5f);
                gl.TexCoord(0.0f, 1.0f);
                gl.Vertex(-0.5f, 0.0f, -0.5f);

                gl.TexCoord(1.0f, 1.0f);
                gl.Vertex(0.5f, 0.0f, -0.5f);
                gl.TexCoord(1.0f, 0.0f);
```

```
                gl.Vertex(0.5f, 0.0f, 0.5f);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(-0.5f, 0.0f, 0.5f);

                //Prednja stranica
                gl.TexCoord(0.5f, 1.0f);
                gl.Vertex(0.0f, 0.8f, 0.0f);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(-0.5f, 0.0f, 0.5f);
                gl.TexCoord(1.0f, 0.0f);
                gl.Vertex(0.5f, 0.0f, 0.5f);

                //Leva stranica
                gl.TexCoord(0.5f, 1.0f);
                gl.Vertex(0.0f, 0.8f, 0.0f);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(-0.5f, 0.0f, -0.5f);
                gl.TexCoord(1.0f, 0.0f);
                gl.Vertex(-0.5f, 0.0f, 0.5f);

                //Zadnja stranica
                gl.TexCoord(0.5f, 1.0f);
                gl.Vertex(0.0f, 0.8f, 0.0f);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(0.5f, 0.0f, -0.5f);
                gl.TexCoord(1.0f, 0.0f);
                gl.Vertex(-0.5f, 0.0f, -0.5f);

                //Desna stranica
                gl.TexCoord(0.5f, 1.0f);
                gl.Vertex(0.0f, 0.8f, 0.0f);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(0.5f, 0.0f, 0.5f);
                gl.TexCoord(1.0f, 0.0f);
                gl.Vertex(0.5f, 0.0f, -0.5f);
                gl.End();
            gl.PopMatrix();

            //Oznaci kraj iscrtavanja
            gl.Flush();
        }

        /// <summary>
        /// Menja trenutno aktivni mod za blending.
        /// </summary>
        public void ChangeTextureBlendMode(OpenGL gl)
        {
            SelectedMode = (TextureBlendingMode)(((int)m_selectedMode
+ 1) % Enum.GetNames(typeof(TextureBlendingMode)).Length);
            switch (m_selectedMode)
```

```
            {
                case TextureBlendingMode.Modulate:
                    gl.TexEnv(OpenGL.GL_TEXTURE_ENV,
OpenGL.GL_TEXTURE_ENV_MODE, OpenGL.GL_MODULATE);
                    break;

                case TextureBlendingMode.Replace:
                    gl.TexEnv(OpenGL.GL_TEXTURE_ENV,
OpenGL.GL_TEXTURE_ENV_MODE, OpenGL.GL_REPLACE);
                    break;

                case TextureBlendingMode.Decal:
                    gl.TexEnv(OpenGL.GL_TEXTURE_ENV,
OpenGL.GL_TEXTURE_ENV_MODE, OpenGL.GL_DECAL);
                    break;

                case TextureBlendingMode.Blend:
                    gl.TexEnv(OpenGL.GL_TEXTURE_ENV,
OpenGL.GL_TEXTURE_ENV_MODE, OpenGL.GL_BLEND);
                    break;

            }
        }
        #endregion

    }
}
```

Listing 9.1 Primer rada sa teksturama - pridruživanje teksture piramidi

Veliki problem predstavlja činjenica što je tekstura fiksnih dimenzija i kao takva teško da se može mapirati na objekat bez potrebe za skaliranjem. Skaliranje skoro uvek daje neželjene artefakte. tzv. *scintillation artefacts*. Ovi artefakti se najbolje uočavaju kod objekata u pokretu. Takođe, skaliranje predstavlja zahtevnu operaciju sa aspekta zauzeća resursa. Kao rešenje za gore navedene probleme koristi se **mipmapping** tehnika pridruživanja teksture objektima. Ova tehnika poboljšava kako performanse iscrtavanja tako i vizuelni kvalitet iscrtanih objekata. *Mipmapping* koristi umesto jedne teksture, čitav niz tekstura (*mipmapping levels)* iste slike, ali različitog kvaliteta i dimenzija. OpenGL tada koristi najpogodniju teksturu. *Mipmapping* tekstura se sastoji od niza slika, gde je svaka slika dva puta manjih dimenzija od prethodne, slika 9.3.

Slika 9.3 *Mipmapping* tekstura

Za kreiranje *mipmapping* tekstura koriste se metode: **gluBuild1DMipmaps**, **gluBuild2DMipmaps** i **gluBuild3DMipmaps**.

```
public static int gluBuild1DMipmaps(int target,
                        int internalFormat,
                        int width,
                        int format,
                        int type,
                        IntPtr data)

public static int gluBuild2DMipmaps(int target,
                        int internalFormat,
                        int width,
                        int height,
                        int format,
                        int type,
                        IntPtr data)
public static int gluBuild3DMipmaps(int target,
                        int internalFormat,
                        int width,
                        int height,
                        int depth,
                        int format,
```

```
                                int type,
                                IntPtr data)
```

gde su:

`target` – odgovarajuća vrednost shodno dimenzionalnosti teksture: *GL_TEXTURE_1D*, *GL_TEXTURE_2D* i *GL_TEXTURE_3D*,
`internalFormat` – format teksela,
`width, height i depth` – dimenzije teksture,
`format, type i data` – su format, tip piksela i pikseli pikselmape koja se koristi kao tekstura.

S obzirom na čestu upotrebu velikog broja tekstura postoji **glGenTextures** metoda koja kreira *N* tekstura odjednom. Ovako kreirana tekstura (se naziva *Texture Object* u OpenGL terminologiji) se pridružuje objektu pozivom metode **glBindTexture**. Teksture se uništavaju pozivom metode **glDeleteTextures**.

```
public static void glGenTextures(int n,
                                  int[] textures)
```

gde su:

`n` – broj tekstura koje treba da se generišu,
`textures` – niz u kom se smeštaju identifikatori alociranih tekstura.

```
public static void glBindTexture(int target,
                                  int texture)
```

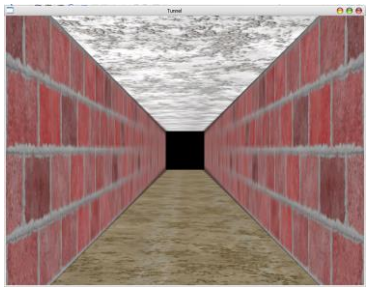
gde su:

`target` –*GL_TEXTURE_1D, GL_TEXTURE_2D i GL_TEXTURE_3D*,
`texture` – identifikator alocirane teksture koja želi koristiti.

```
public static void glDeleteTextures(int n,
                                     int[] textures)
```

gde su:

`n` – broj tekstura koje treba da se unište,
`textures` – niz identifikatora tekstura.



Slika 9.4 Rezultat primera iz listinga 9.2

Listing 9.2 i slika 9.4 prikazuju primer korišćenja gore navedenih metoda. Primer pokazuje primenu *mipmapping* tekstura, kao i primenu različitih filtara nad teksturama.

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using SharpGL;

namespace MipMaps{
    /// <summary>
    ///  Nabrojani tip OpenGL rezima filtriranja tekstura
    /// </summary>
    public enum TextureFilterMode
    {
        Nearest,
        Linear,
        NearestMipmapNearest,
        NearestMipmapLinear,
        LinearMipmapNearest,
        LinearMipmapLinear
    };

    /// <summary>
    ///  Klasa enkapsulira OpenGL kod i omogucava njegovo iscrtavanje
i azuriranje.
    /// </summary>
    public class World : IDisposable
    {
        #region Atributi

        /// <summary>
        ///    Identifikatori tekstura za jednostavniji pristup
teksturama
        /// </summary>
        private enum TextureObjects { Brick = 0, Floor, Ceiling };
```

```
        private readonly int m_textureCount =
Enum.GetNames(typeof(TextureObjects)).Length;

        /// <summary>
        ///    Identifikatori OpenGL tekstura
        /// </summary>
        private uint[] m_textures = null;

        /// <summary>
        ///    Putanje do slika koje se koriste za teksture
        /// </summary>
        private string[] m_textureFiles = {
"..//..//images//brick.jpg", "..//..//images//floor.jpg",
"..//..//images//ceiling.jpg" };

        /// <summary>
        ///  Izabrana OpenGL mehanizam za iscrtavanje.
        /// </summary>
        private TextureFilterMode m_selectedMode =
TextureFilterMode.Nearest;

        /// <summary>
        ///  Pomeraj po Z osi
        /// </summary>
        private float m_zPosition = -60.0f;

        /// <summary>
        ///    Sirina OpenGL kontrole u pikselima.
        /// </summary>
        private int m_width;

        /// <summary>
        ///    Visina OpenGL kontrole u pikselima.
        /// </summary>
        private int m_height;

        /// <summary>
        ///    Referenca na OpenGL instancu unutar aplikacije.
        /// </summary>
        private OpenGL gl;

        #endregion Atributi

        #region Properties

        /// <summary>
        ///  Izabrani OpenGL rezim stapanja tekstura sa materijalom
        /// </summary>
        public TextureFilterMode SelectedMode
        {
```

```
            get { return m_selectedMode; }
            set
            {
                m_selectedMode = value;

                foreach (uint textureId in m_textures)
                {
                    gl.BindTexture(OpenGL.GL_TEXTURE_2D, textureId);

                    switch (m_selectedMode)
                    {
                        case TextureFilterMode.Nearest:
                            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_NEAREST);
                            break;

                        case TextureFilterMode.Linear:
                            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_LINEAR);
                            break;

                        case TextureFilterMode.NearestMipmapNearest:
                            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_NEAREST_MIPMAP_NEAREST);
                            break;

                        case TextureFilterMode.NearestMipmapLinear:
                            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_NEAREST_MIPMAP_LINEAR);
                            break;

                        case TextureFilterMode.LinearMipmapNearest:
                            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_LINEAR_MIPMAP_NEAREST);
                            break;

                        case TextureFilterMode.LinearMipmapLinear:
                            gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_LINEAR_MIPMAP_LINEAR);
                            //gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MAG_FILTER, OpenGL.GL_LINEAR_MIPMAP_LINEAR);
                            break;
                    }
                }
            }
        }

        /// <summary>
        ///    Ugao rotacije sveta oko X ose.
        /// </summary>
```

```
        public float PositionZ
        {
            get { return m_zPosition; }
            set { m_zPosition = value; }
        }

        /// <summary>
        ///    Sirina OpenGL kontrole u pikselima.
        /// </summary>
        public int Width
        {
            get { return m_width; }
            set { m_width = value; }
        }

        /// <summary>
        ///    Visina OpenGL kontrole u pikselima.
        /// </summary>
        public int Height
        {
            get { return m_height; }
            set { m_height = value; }
        }

        #endregion Properties

        #region Konstruktori

        /// <summary>
        ///  Konstruktor.
        /// </summary>
        public World(OpenGL gl)
        {
            this.gl = gl;
            m_textures = new uint[m_textureCount];
        }

        #endregion Konstruktori
       #region Metode

        /// <summary>
        ///  Korisnicka inicijalizacija i podesavanje OpenGL
parametara.
        /// </summary>
        public void Initialize()
        {
            // Crna pozadina
            gl.ClearColor(0.0f, 0.0f, 0.0f, 1.0f);
```

```
            // Ukljuci depth testing i back face culling i podesi da
je front = CW
            gl.Enable(OpenGL.GL_DEPTH_TEST);
            gl.Enable(OpenGL.GL_CULL_FACE);
            gl.FrontFace(OpenGL.GL_CW);

            // Teksture se primenjuju sa parametrom decal
            gl.Enable(OpenGL.GL_TEXTURE_2D);
            gl.TexEnv(OpenGL.GL_TEXTURE_ENV,
OpenGL.GL_TEXTURE_ENV_MODE, OpenGL.GL_DECAL);

            // Ucitaj slike i kreiraj teksture
            gl.GenTextures(m_textureCount, m_textures);
            for (int i = 0; i < m_textureCount; ++i)
            {
                // Pridruzi teksturu odgovarajucem identifikatoru
                gl.BindTexture(OpenGL.GL_TEXTURE_2D, m_textures[i]);

                // Ucitaj sliku i podesi parametre teksture
                Bitmap image = new Bitmap(m_textureFiles[i]);
                // rotiramo sliku zbog koordinantog sistema opengl-a
                image.RotateFlip(RotateFlipType.RotateNoneFlipY);
                Rectangle rect = new Rectangle(0, 0, image.Width,
image.Height);
                // RGBA format (dozvoljena providnost slike tj. alfa
kanal)
                BitmapData imageData = image.LockBits(rect,
System.Drawing.Imaging.ImageLockMode.ReadOnly,

System.Drawing.Imaging.PixelFormat.Format32bppArgb);

                gl.Build2DMipmaps(OpenGL.GL_TEXTURE_2D,
(int)OpenGL.GL_RGBA8, image.Width, image.Height, OpenGL.GL_BGRA,
OpenGL.GL_UNSIGNED_BYTE, imageData.Scan0);
                gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_LINEAR);       // Linear
Filtering
                gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MAG_FILTER, OpenGL.GL_LINEAR);       // Linear
Filtering

                image.UnlockBits(imageData);
                image.Dispose();
            }
        }

        /// <summary>
        /// Podesava viewport i projekciju za OpenGL kontrolu.
        /// </summary>
        public void Resize(int width, int height)
```

```
        {
            m_width = width;
            m_height = height;
            gl.Viewport(0, 0, m_width, m_height);
            gl.MatrixMode(OpenGL.GL_PROJECTION);
            gl.LoadIdentity();
            gl.Perspective(90.0, (double)m_width / m_height, 1.0,
120.0);
            gl.MatrixMode(OpenGL.GL_MODELVIEW);
            gl.LoadIdentity();
        }

        /// <summary>
        ///  Iscrtavanje OpenGL kontrole.
        /// </summary>
        public void Draw()
        {
            gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT |
OpenGL.GL_DEPTH_BUFFER_BIT);
            gl.PushMatrix();
            // Pomeraj objekat po z-osi
            gl.Translate(0.0f, 0.0f, m_zPosition);

            for (float z = 60.0f; z >= 0.0f; z -= 10.0f)
            {
                // Pod tunela
                gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)TextureObjects.Floor]);
                gl.Begin(OpenGL.GL_QUADS);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(-10.0f, -10.0f, z);
                gl.TexCoord(0.0f, 1.0f);
                gl.Vertex(-10.0f, -10.0f, z - 10.0f);
                gl.TexCoord(1.0f, 1.0f);
                gl.Vertex(10.0f, -10.0f, z - 10.0f);
                gl.TexCoord(1.0f, 0.0f);
                gl.Vertex(10.0f, -10.0f, z);
                gl.End();

                // Plafon tunela
                gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)TextureObjects.Ceiling]);
                gl.Begin(OpenGL.GL_QUADS);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(-10.0f, 10.0f, z);
                gl.TexCoord(1.0f, 0.0f);
                gl.Vertex(10.0f, 10.0f, z);
                gl.TexCoord(1.0f, 1.0f);
                gl.Vertex(10.0f, 10.0f, z - 10.0f);
                gl.TexCoord(0.0f, 1.0f);
```

```
                gl.Vertex(-10.0f, 10.0f, z - 10.0f);
                gl.End();

                // Levi zid tunela
                gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)TextureObjects.Brick]);
                gl.Begin(OpenGL.GL_QUADS);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(-10.0f, -10.0f, z);
                gl.TexCoord(0.0f, 1.0f);
                gl.Vertex(-10.0f, 10.0f, z);
                gl.TexCoord(1.0f, 1.0f);
                gl.Vertex(-10.0f, 10.0f, z - 10.0f);
                gl.TexCoord(1.0f, 0.0f);
                gl.Vertex(-10.0f, -10.0f, z - 10.0f);
                gl.End();

                // Desni zid tunela
                gl.Begin(OpenGL.GL_QUADS);
                gl.TexCoord(0.0f, 0.0f);
                gl.Vertex(10.0f, -10.0f, z);
                gl.TexCoord(1.0f, 0.0f);
                gl.Vertex(10.0f, -10.0f, z - 10.0f);
                gl.TexCoord(1.0f, 1.0f);
                gl.Vertex(10.0f, 10.0f, z - 10.0f);
                gl.TexCoord(0.0f, 1.0f);
                gl.Vertex(10.0f, 10.0f, z);
                gl.End();
            }

            gl.PopMatrix();
            // Oznaci kraj iscrtavanja
            gl.Flush();
        }

        /// <summary>
        /// Menja trenutno aktivni mod za filtering.
        /// </summary>
        public void ChangeTextureFilterMode()
        {
            SelectedMode = (TextureFilterMode)(((int)m_selectedMode +
1) % Enum.GetNames(typeof(TextureFilterMode)).Length);
        }

        #endregion Metode

        #region IDisposable metode

        /// <summary>
        ///  Dispose metoda.
```

```
        /// </summary>
        public void Dispose()
        {
            this.Dispose(true);
            GC.SuppressFinalize(this);
        }

        /// <summary>
        ///  Destruktor.
        /// </summary>
        ~World()
        {
            this.Dispose(false);
        }

        /// <summary>
        ///  Implementacija IDisposable interfejsa.
        /// </summary>
        protected virtual void Dispose(bool disposing)
        {
            if (disposing)
            {
                gl.DeleteTextures(m_textureCount, m_textures);
            }
        }

        #endregion IDisposable metode
    }
}
```

Listing 9.2 Primer korišćenja *mipmapping* tekstura

Uobičajno je da se teksture koriste za simulaciju realnog okruženja (pozadine scene). Listing 9.3 i slika 9.5 prikazuju primer definisanja pozadine scene – korišćenjem kvadra čijim stranicama su pridružene teksture pozadine.

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using SharpGL;

namespace GenTexCoord
{
    /// <summary>
    ///  Klasa enkapsulira OpenGL kod i omogucava njegovo iscrtavanje
i azuriranje.
    /// </summary>
    public class World
    {
```

```
        #region Atributi

        /// <summary>
        ///   Ugao rotacije kamere oko Y ose.
        /// </summary>
        private float m_angle = 0.0f;

        /// <summary>
        ///    Sirina OpenGL kontrole u pikselima.
        /// </summary>
        private int m_width;

        /// <summary>
        ///    Visina OpenGL kontrole u pikselima.
        /// </summary>
        private int m_height;

        /// <summary>
        ///    Identifikatori tekstura za bolju citljivost.
        /// </summary>
        private enum TextureObjects { Back = 0, Front, Bottom, Top,
Left, Right };

        /// <summary>
        ///    Identifikatori OpenGL tekstura
        /// </summary>
        private uint[] m_textures = null;

        /// <summary>
        ///    Broj tekstura.
        /// </summary>
        private readonly int m_textureCount =
Enum.GetNames(typeof(TextureObjects)).Length;

        /// <summary>
        ///    Referenca na OpenGL instancu u aplikaciji.
        /// </summary>
        private OpenGL gl;

        #endregion Atributi

        #region Properties

        public float Angle
        {
            get { return m_angle; }
            set { m_angle = value; }
        }

        /// <summary>
```

```
        ///    Sirina OpenGL kontrole u pikselima.
        /// </summary>
        public int Width
        {
            get { return m_width; }
            set { m_width = value; }
        }

        /// <summary>
        ///    Visina OpenGL kontrole u pikselima.
        /// </summary>
        public int Height
        {
            get { return m_height; }
            set { m_height = value; }
        }

        #endregion Properties

        #region Konstruktori

        /// <summary>
        ///  Konstruktor.
        /// </summary>
        public World(OpenGL gl)
        {
            this.gl = gl;
            m_textures = new uint[m_textureCount];
        }

        #endregion Konstruktori

        #region Metode

        /// <summary>
        ///  Korisnicka inicijalizacija i podesavanje OpenGL
parametara.
        /// </summary>
        public void Initialize()
        {
            gl.ClearColor(0.0f, 0.0f, 0.0f, 1.0f);

            gl.Enable(OpenGL.GL_TEXTURE_2D);
            gl.Enable(OpenGL.GL_DEPTH_TEST);

            CreateTextures();
        }

        /// <summary>
        /// Podesava viewport i projekciju za OpenGL kontrolu.
```

```
        /// </summary>
        public void Resize(int width, int height)
        {
            m_width = width;
            m_height = height;
            gl.Viewport(0, 0, m_width, m_height);
            gl.MatrixMode(OpenGL.GL_PROJECTION);
            gl.LoadIdentity();
            gl.Perspective(45.0, (double)m_width / (double)m_height,
0.1, 500.0);
            gl.MatrixMode(OpenGL.GL_MODELVIEW);
            gl.LoadIdentity();
        }

        /// <summary>
        ///  Iscrtavanje OpenGL kontrole.
        /// </summary>
        public void Draw()
        {
            gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT |
OpenGL.GL_DEPTH_BUFFER_BIT);
            gl.MatrixMode(OpenGL.GL_MODELVIEW);
            gl.LoadIdentity();

            gl.Rotate(0f, m_angle, 0f);
            // nacrtaj okolinu
            DrawEnviroment(0.0f, 0.0f, 0.0f, 400.0f, 200.0f, 400.0f);

            // Oznaci kraj iscrtavanja
            gl.Flush();
        }
 /// <summary>
        ///  Iscrtavanje teksturiranog kvadra - okruzenja.
        /// </summary>
        private void DrawEnviroment(float x, float y, float z, float
width, float height, float length)
        {
            // BACK teksturu pridruzi zadnjoj stranici kocke
            gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)TextureObjects.Back]);

            // kocka je centrirana oko (x,y,z) tacke
            x = x - width / 2;
            y = y - height / 2;
            z = z - length / 2;

            // BACK stranica
            gl.Begin(OpenGL.GL_QUADS);
            gl.TexCoord(1.0f, 0.0f);
            gl.Vertex(x + width, y, z);
```

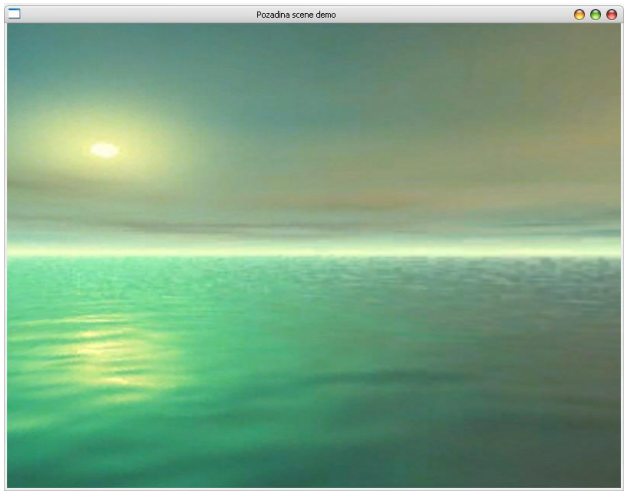
```
            gl.TexCoord(1.0f, 1.0f);
            gl.Vertex(x + width, y + height, z);
            gl.TexCoord(0.0f, 1.0f);
            gl.Vertex(x, y + height, z);
            gl.TexCoord(0.0f, 0.0f);
            gl.Vertex(x, y, z);
            gl.End();

            // FRONT teksturu pridruzi prednjoj stranici kocke
            gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)TextureObjects.Front]);

            // FRONT stranica
            gl.Begin(OpenGL.GL_QUADS);
            gl.TexCoord(1.0f, 0.0f);
            gl.Vertex(x, y, z + length);
            gl.TexCoord(1.0f, 1.0f);
            gl.Vertex(x, y + height, z + length);
            gl.TexCoord(0.0f, 1.0f);
            gl.Vertex(x + width, y + height, z + length);
            gl.TexCoord(0.0f, 0.0f);
            gl.Vertex(x + width, y, z + length);
            gl.End();

            // BOTTOM teksturu pridruzi donjoj stranici kocke
            gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)TextureObjects.Bottom]);

            // BOTTOM stranica
            gl.Begin(OpenGL.GL_QUADS);
            gl.TexCoord(1.0f, 0.0f); gl.Vertex(x, y, z);
            gl.TexCoord(1.0f, 1.0f); gl.Vertex(x, y, z + length);
            gl.TexCoord(0.0f, 1.0f); gl.Vertex(x + width, y, z +
length);
            gl.TexCoord(0.0f, 0.0f); gl.Vertex(x + width, y, z);
            gl.End();

            // TOP teksturu pridruzi gornjoj stranici
            gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)TextureObjects.Top]);

            // TOP stranica
            gl.Begin(OpenGL.GL_QUADS);
            gl.TexCoord(0.0f, 1.0f); gl.Vertex(x + width, y + height,
z);
            gl.TexCoord(0.0f, 0.0f); gl.Vertex(x + width, y + height,
z + length);
            gl.TexCoord(1.0f, 0.0f); gl.Vertex(x, y + height, z +
length);
            gl.TexCoord(1.0f, 1.0f); gl.Vertex(x, y + height, z);
```

```
            gl.End();

            // LEFT teksturu pridruzi levoj stranici
            gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)TextureObjects.Left]);

            // LEFT stranica
            gl.Begin(OpenGL.GL_QUADS);
            gl.TexCoord(1.0f, 1.0f); gl.Vertex(x, y + height, z);
            gl.TexCoord(0.0f, 1.0f); gl.Vertex(x, y + height, z +
length);
            gl.TexCoord(0.0f, 0.0f); gl.Vertex(x, y, z + length);
            gl.TexCoord(1.0f, 0.0f); gl.Vertex(x, y, z);
            gl.End();

            // RIGHT teksturu pridruzi desnoj stranici
            gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)TextureObjects.Right]);

            // RIGHT stranica
            gl.Begin(OpenGL.GL_QUADS);
            gl.TexCoord(0.0f, 0.0f); gl.Vertex(x + width, y, z);
            gl.TexCoord(1.0f, 0.0f); gl.Vertex(x + width, y, z +
length);
            gl.TexCoord(1.0f, 1.0f); gl.Vertex(x + width, y + height,
z + length);
            gl.TexCoord(0.0f, 1.0f); gl.Vertex(x + width, y + height,
z);
            gl.End();
        }

        /// <summary>
        ///  Kreiranje teksture.
        /// </summary>
        private void CreateTextures()
        {
            string[] textureFiles = new string[] {
"..//..//images//back.jpg",

"..//..//images//front.jpg",

"..//..//images//bottom.jpg",

"..//..//images//top.jpg",

"..//..//images//left.jpg",

"..//..//images//right.jpg" };

            gl.GenTextures(m_textureCount, m_textures);
```

```
            gl.TexEnv(OpenGL.GL_TEXTURE_ENV,
OpenGL.GL_TEXTURE_ENV_MODE, OpenGL.GL_DECAL);

            for (int textureID = 0; textureID < m_textures.Length;
textureID++)
            {
                Bitmap image = new Bitmap(textureFiles[textureID]);
                image.RotateFlip(RotateFlipType.RotateNoneFlipY);
                Rectangle rect = new Rectangle(0, 0, image.Width,
image.Height);
                BitmapData bitmapdata = image.LockBits(rect,
System.Drawing.Imaging.ImageLockMode.ReadOnly,

System.Drawing.Imaging.PixelFormat.Format32bppArgb);

                gl.BindTexture(OpenGL.GL_TEXTURE_2D,
m_textures[(int)textureID]);

                gl.Build2DMipmaps(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_RGBA8, image.Width, image.Height, OpenGL.GL_BGRA,
OpenGL.GL_UNSIGNED_BYTE, bitmapdata.Scan0);

                gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MAG_FILTER, OpenGL.GL_LINEAR);
                gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_LINEAR_MIPMAP_LINEAR);
                gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_WRAP_S, OpenGL.GL_CLAMP_TO_EDGE);
                gl.TexParameter(OpenGL.GL_TEXTURE_2D,
OpenGL.GL_TEXTURE_WRAP_T, OpenGL.GL_CLAMP_TO_EDGE);

                image.UnlockBits(bitmapdata);
                image.Dispose();
            }
        }


        /// <summary>
        ///  Dispose metoda.
        /// </summary>
        public void Dispose()
        {
            this.Dispose(true);
            GC.SuppressFinalize(this);
        }

        /// <summary>
        ///  Destruktor.
        /// </summary>
        ~World()
```

```
        {
            this.Dispose(false);
        }

        #endregion Metode

        #region IDisposable metode

        /// <summary>
        ///  Implementacija IDisposable interfejsa.
        /// </summary>
        protected virtual void Dispose(bool disposing)
        {
            if (disposing)
            {
                Terminate();
            }
        }

        /// <summary>
        ///  Korisnicko oslobadjanje OpenGL resursa.
        /// </summary>
        private void Terminate()
        {
            gl.DeleteTextures(m_textureCount, m_textures);
        }

        #endregion IDisposable metode
    }
}
```

Listing 9.3 Primer definisanja pozadine scene



Slika 9.5 Rezultat primera iz listinga 9.3

LITERATURA

1. GLUT specification, http://www.opengl.org/resources/libraries/glut/glut-3.spec.pdf
2. OpenGL specification, http://www.opengl.org/documentation/specs/
3. Wright R. Jr., Lipchak B., Haemel N., „OpenGL Super Bible“, 4th Ed., Addison Wesley, 2007.
4. Spherical coordinates, http://mathworld.wolfram.com/SphericalCoordinates.html (Gasiorowicz 1974, pp. 167-168; Arfken 1985, p. 108).