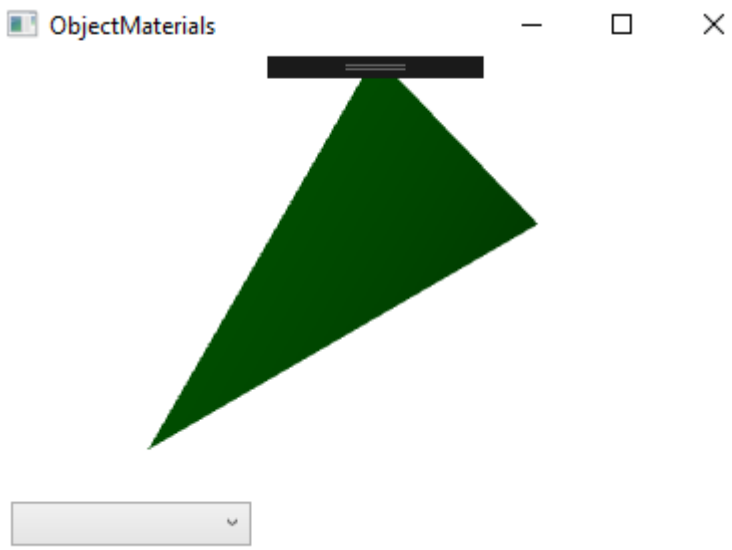
Teksture, materijali, 3D površine, 3D modeli, animacija 3D objekata

Skripta objašnjenja primera za gradivo 10. nedelje:

- **teksture**,
- **materijali – obnavljanje,**
- **3D površine i materijali,**
- **animacija 3D objekata,**
- **interakcija**

U Visual Studio projektu Primer_3.1, prvo će biti objašnjen sadržaj prozora **ObjectMaterials.xaml** (prikazan na **slici 1**).



Slika 1 - Sadržaj prozora ObjectMaterials.xaml

Na prozoru je prikazan jedan *mesh* model, sastavljen od samo jednog trougla. Podešena je perspektivna kamera, dok je za izvor svetlosti odabrano reflektorsko svetlo, odnosno klasa ***SpotLight***. Na dnu prozora se nalazi ***ComboBox***, preko kojeg je moguće izabrati materijal iscrtanog modela.

Kao što je objašnjeno u skripti za gradivo vežbi 9. nedelje, materijal modela se opisuje preko svojih refleksivnih karakteristika. Moguće ga je opisati preko tri svetlosne komponente: difuzne, emisione i spekularne. Kako je svaka od ovih komponenti detaljno opisana u prethodnoj skripti, o njihovim karakteristikama ovde više neće biti reči.

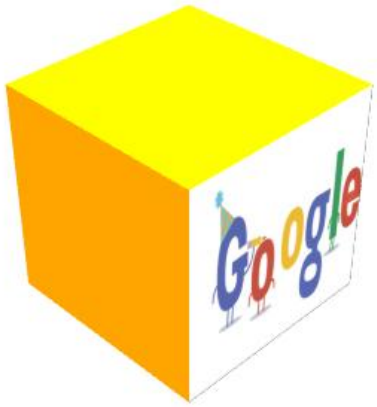
Materijal *mesh* modela se definiše definisanjem boja pojedinačnih svetlosnih komponenti materijala. Ukoliko se podešava samo jedna komponenta materijala, dovoljno je instancirati novi objekat klase tog materijala (moguće opcije su: ***DiffuseMaterial***, ***SpecularMaterial*** i ***EmmisiveMaterial***), pa zatim odrediti boju tog materijala koristeći ***Brush*** željene boje.

U slučaju da je potrebno definisati više (ili sve) komponenti materijala, mora se pre toga instancirati objekat klase ***MaterialGroup***. Ova klasa poseduje listu nazvanu ***Children***, u koju je potrebno ubaciti svaki novopodešeni materijal. Na **listingu 1** je prikazano podešavanje difuzne svetlosne komponente materijala na crnu boju i emisione komponente na zelenu. Postupak je u potpunosti analogan i za podešavanje spekularne komponente.

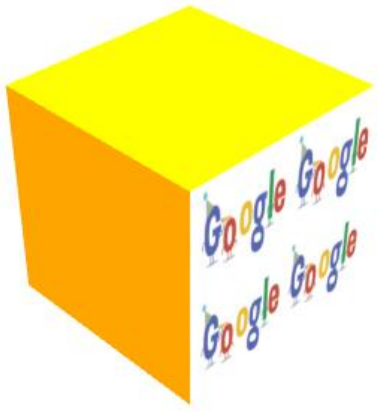
```
case "Diffuse - Black, Emissive - Green":
    MaterialGroup mg1 = new MaterialGroup();
    DiffuseMaterial dm2 = new DiffuseMaterial();
    dm2.Brush = Brushes.Black;
    mg1.Children.Add(dm2);
    EmissiveMaterial em = new EmissiveMaterial();
    em.Brush = Brushes.Green;
    mg1.Children.Add(em);
    myModel.Material = mg1;
    break;
```

Listing 1 - podešavanje difuzne i emisione komponente materijala

Teksture će biti objašnjene na primeru sadržaja prozora **ObjectTextures.xaml**. Pridruživanje teksture objektima doprinosi realizmu scene. Za razliku od materijala koji samo odražavaju refleksivna svojstva površine objekta, tekstura definiše izgled površine. Moguće je pridružiti sliku (npr JPEG) teksturi, koja će biti prikazana na površini objekta. Primer korišćenja tekstura je prikazan na **slikama 2 i 3**.



Slika 2 - primer korišćenja teksture

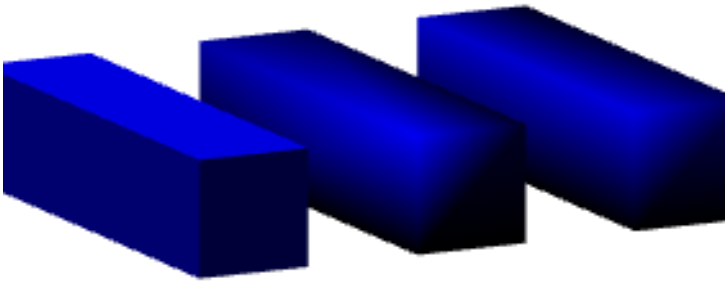


Slika 3 - primer ponavljanja teksture

Podešavanje teksture (**slika 2**) se svodi na definisanje ***Brush-a*** difuzne komponente materijala objekta. U ovom slučaju se ne bira boja, već se, korišćenjem objekta ***ImageBrush*** klase određuje koja slika će biti prikazana na površini objekta.

Ukoliko je potrebno ponoviti teksturu (**slika 3**), potrebno je dodatno podesiti *property* ***TileMode*** na *Tile* i definisati *property* ***Viewbox***, odnosno način ponavljanja iscrtane teksture. Takođe, potrebno je podesiti koordinate tekstura, podešavanjem *property*-a ***TextureCoordinates*** unutar objekta klase ***MeshGeometry3D***, koji je zadužen za opisivanje geometrije 3D objekta.

Senčenje će biti objašnjeno na primeru sadržaja prozora **Shading.xaml**, koji je prikazan na **slici 4**. Na slici su prikazana tri *mesh* modela, odnosno tri kvadra. Kod sva tri modela je boja difuzne svetlosne komponente podešena na plavu. Na sceni je postavljena kamera sa ortogonalnom projekcijom. Podešen je direkcioni izvor svetlosti bele boje sa smerom osvetljaja prema 3D vektoru sa koordinatama (2, -4, -1).



Slika 4 - Sadržaj prozora Shading.xaml

Na prozoru se da primetiti da je senčenje na prvom kvadru sa leve strane najjednostavnije, dok postaje složenije na drugom i na trećem kvadru. Razlog tome je što se svaki sledeći kvadar sastoji od više trouglova, a samim tim i više normala. Kako normala svake površine utiče na to kako se reflektuje svetlost sa te površine, logičan je zaključak da će refleksija svetlosti, a samim tim i njeno odsustvo, tj senka, biti kvalitetniji ukoliko se objekat sastoji od više trouglova.

Zumiranje objekta na sceni će biti objašnjeno na primeru sadržaja prozora **ZoomPan.xaml**. Na prozoru je prikazan *mesh* objekat (kvadrat), sastavljen od dva trougla, sa dodatom teksturom. Takođe je podešen 3D ***Viewport*** i dodata kamera sa perspektivnom projekcijom, kako samo zumiranje imalo smisla.

Efekat zumiranja se postiže pomoću transformacije skaliranja. Pri pokretanju programa je skaliranje po svakoj od osa (***ScaleX***, ***ScaleY*** i ***ScaleZ***) podešeno na 1 (osnovna veličina modela). Da bi se postigao efekat zumiranja potrebno je postaviti *listener* na događaj pomeranja "točkića" miša, odnosno korisničke akcije *scroll*-a, kao što je prikazano na **listingu 2**.

Prilikom aktiviranja *listener*-a na koju stranu je pomeren "točkić", odnosno da li je pokušan uvećavanje ili smanjivanje objekta. To se postiže proveravanjem vrednosti polja ***Delta*** objekta klase ***MouseWheelEventArgs***. Ukoliko je ta vrednost veća od 0, pokušava se uvećanje, odnosno zumiranje objekta, i obrnuto. Takođe je potrebno proveriti da li vrednost trenutnog zuma (u kodu ***zoomCurrent***) prevazilaze predefinisane granice (u kodu ***zoomMax*** za maksimalno uvećanje i – ***zoomMax*** za maksimalno umanjenje).

Prilikom aktiviranja *listener*-a, uz pretpostavku da su zadovoljeni prethodni uslovi, vrednosti skaliranja po osama X i Y se uvećavaju i umanjuju za 0.1, analogno tome se i vrednost trenutnog zuma menja za 1.

```
private void viewport1_MouseWheel(object sender, MouseWheelEventArgs e)
{
    Point p = e.MouseDevice.GetPosition(this);
    double scaleX = 1;
    double scaleY = 1;
    if (e.Delta > 0 && zoomCurent<zoomMax)
    {
        scaleX = skaliranje.ScaleX + 0.1;
        scaleY = skaliranje.ScaleY + 0.1;
        zoomCurent++;
        skaliranje.ScaleX = scaleX;
        skaliranje.ScaleY = scaleY;
    }
    else if(e.Delta <= 0 && zoomCurent > -zoomMax)
    {
        scaleX = skaliranje.ScaleX - 0.1;
        scaleY = skaliranje.ScaleY - 0.1;
        zoomCurent--;
        skaliranje.ScaleX = scaleX;
        skaliranje.ScaleY = scaleY;
```

Listing 2 - Primer zumiranja

Analogno zumiranju je implementirano i pomeranje objekata prevlačenjem miša. Napravljen je *listener* pomeraja miša uz uslov da je za to vreme aktivan i levi klik. Translacija objekta po X i Y osi se preračunava u odnosu na pomeraj miša (**listing 3**).

```
private void viewport1_MouseMove(object sender, MouseEventArgs e)
{
    if (viewport1.IsMouseCaptured)
    {
        Point end = e.GetPosition(this);
        double offsetX = end.X - start.X;
        double offsetY = end.Y - start.Y;
        double w = this.Width;
        double h = this.Height;
        double translateX = (offsetX * 100) / w;
        double translateY = -(offsetY * 100) / h;
        translacija.OffsetX = diffOffset.X + (translateX / (100*skaliranje.ScaleX));
        translacija.OffsetY = diffOffset.Y + (translateY / (100*skaliranje.ScaleX));
    }
}
```

Listing 3 - Primer translacije

WPF grafika omogućava i učitavanja gotovih 3D modela. Način na koji se to postiže biće opisan na primeru sadržaja prozora **ImportModel.xaml** (slika 5)**.** Za potrebe ovog primera korišćena je biblioteka ***HelixToolkit*** .

Da bi se uspešno učitao 3D model, potrebno je prvo definisati *Viewport* iz ***HelixToolkit*** biblioteke, odnosno koristiti objekat klase ***HelixViewport3D***. Korisni *property*-i:

- ***ZoomExtentsWhenLoaded*** – tipa bool, ako se postavi na *true*, omogućava automatsko podešavanje zuma prozora tako da se prikaže ceo model, ukoliko je on veći od trenutnog zuma prozora,
- ***ShowCameraTarget*** – tipa bool, ako se postavi na *true*, prikazuje ikonu koja ukazuje na smer vektora u kom je kamera okrenuta,
- ***ShowViewCube*** – tipa bool, ako se postavi na *true*, prikazuje ***ViewCube*** u donjem desnom uglu prozora, koji ukazuje na to koja strana učitanog modela se prikazuje.

Takođe je moguće podesiti rotiranje modela korišćenjem miša. Potrebno je definisati novi objekat klase ***MouseGesture***, čiji konstruktor prihvata za parametar akciju miša, i takav objekat pridružiti *ViewPort*-u. Za osvetljenje *Viewport*-a korišćeno je ambijentalno osvetljenje ***HelixToolkit*** biblioteke (*<helix:SunLight/>*).

Da bi se model uspešno učitao u *Viewport*, potrebno je prvo instancirati novi objekat klase ***ModelVisual3D***, čije će polje ***Content*** sadržati model, kada se on jednom učita. Sam model se učitava uz pomoć klase ***ModelImporter***, korišćenjem njene metode *Load(string path)*. Ova metoda za povratnu vrednost ima objekat klase ***Model3DGroup***, čiji tip odgovara gorepomenutom polju ***Content***.

Ovako učitan model se iz objekta ***ModelVisual3D*** prosleđuje objektu klase ***ModelUIElement3D*** (njenom polju ***Model***), da bi se prikazao na ekranu.

Modelu je dodat materijal sa belom difuznom komponentom, kako on ne bi bio podrazumevano obojen istom bojom kao pozadina.

Slika 5 - Sadržaj prozora ImportModel.xaml

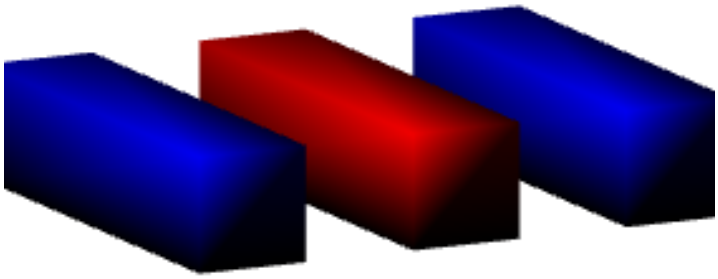
Postupak za učitavanje modela je isti za modele bez tekstura, kao i za modele sa teksturama (kod modela sa teksturama nije potrebno definisati materijal, jer je on već podešen u modelu).

Animacija 3D objekata će biti objašnjena na primeru sadržaja prozora **Animation3D.xaml**. Na prozoru je prikazan *mesh* model kvadra sastavljenog od 12 trouglova. Podešena je kamera sa perspektivnom projekcijom i direkcioni izvor svetlosti bele boje. Kvadar je animiran da se se okreće oko ose sa koordinatama (2, 1, 0).

Sama animacija je implementirana korišćenjem ***Trigger***-a koji pripadaju objektu klase ***ViewPort3D***. Da bi se animacija uspešno izvela potrebno je podesiti nekoliko parametara:

- ***RoutedEvent*** – *property* objekta klase ***EventTrigger***, određuje događaj koji će biti okidač za pokretanje animacije. U ovom slučaju je odabran dogadjaj učitavanja *Viewport*-a,
- ***Storyboard*** – opisuje samu animaciju, potrebno je instancirati objekat ove klase unutar objekta klase ***BeginStoryboard***,
- ***Storyboard.TargetName*** – *property* koji povezuje animaciju sa objektom koji je potrebno animirati, potrebno je uneti njegovo ime (u ovom primeru je to *x:Name="myAngleRotation"*),
- ***Storyboard.TargetProperty*** **–** property kojim se određuje tačan *property* ciljanog objekta na koji će animacija uticati,
- ***RepeatBehavior*** – način ponavljanja animacije, u ovom primeru je podešeno na beskonačno ponavljanje
- property-i **From**, **To** i **Duration –** određuju, respektivno, početnu i krajnju vrednost ciljanog *property*-a, dužinu trajanja animacije.

U Visual Studio projektu Primer3_2 će biti objašnjena korisnička interakcija sa 3D scenom, na primeru prozora **HitTestWPF.xaml** (**slika 6**).



Slika 6 - Sadržaj prozora HitTestWPF.xaml

Na slici su prikazana tri mesh modela, odnosno tri kvadra. Kod sva tri modela je boja difuzne svetlosne komponente podešena na plavu. Na sceni je postavljena kamera sa ortogonalnom projekcijom. Podešen je direkcioni izvor svetlosti bele boje sa smerom osvetljaja prema 3D vektoru sa koordinatama (2, -4, -1).

Prvobitno su sva tri kvadra iscrtana u plavoj boji. Klikom miša na nekog od njih, kliknuti kvadar postaje crven. Ukoliko se klikne na neki od preostala dva, novokliknuti kvadar postaje crven, dok se prethodni kvadar ponovo boji u plavu boju.

Klikom miša bilo gde na *Viewport*-u aktivira se ***MouseDown*** *listener*. Tada se instanciraju objekti klasa ***Point3D*** (promenljiva imenovana *testpoint3D*) i ***Vector3D*** (promenljiva imenovana *testdirection*). U *testpoint3D* se smešta trenutni položaj pokazivača miša, dok se u *testdirection* smešta vektor pravca u kom pokazivač pokazuje (radi se u 3D prostoru, mora se uzeti u obzir i dubina scene).

Za pronalaženje kliknutog elementa koristi se metoda ***VisualTreeHelper.HitTest()***. Podsetnik sa početka kursa: Svaki iscrtani objekat se smešta u internu strukturu stabla, stoga korišćenje klase ***VisualTreeHelper***.

Ova metoda za parametre prima, između ostalog, referencu na vizuelnu komponentu unutar koje je potrebno izvršiti traženje kliknutog elementa (hit hest), poziv redefinisane metode ***HitTestResultBehavior HTResult*** kojom se definiše šta će se dogoditi jednom kada se kliknuti objekat pronađe i objekat klase ***HitTestParameteres*** koja sadrži sve informacije o tome gde se kliknulo na prozoru. U ovom slučaju iskorišćena je njena klasa naslednica – ***PointHitTestParameters***.

Kada se jednom pronađe kliknuti model, njegova boja difuzne komponente menja se u crvenu.

Na samom prozoru je definisan je definisan još jedan *listener*, i to ***MouseLeftButtonDown***. Pri njegovom aktiviranju se svi modeli unutar prozora boje u plavu boju. Ovim se postiže da se svakim narednim klikom na model prethodno kliknuti model vrati u prvobitno stanje. Tek nakon aktiviranja ovog *listener*-a se aktivira prethodno opisani ***MouseDown*** koji osluškuje događaje na *Viewport*-u.