Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

OpenGL / WebGL

Marko Čupić

Zagreb, 11. travnja 2019.

OpenGL / WebGL

- Tehnologija koja omogućava prikaz trodimenzijskih scena učinkovitom uporabom grafičkih kartica
- OpenGL – namijenjen razvoju nativnih aplikacija
- WebGL – podskup OpenGL-a koji se može izravno koristiti u Web-pregledniku za renderiranje 3D-scena

OpenGL / WebGL

- OpenGL
  - v 1.0: 1992.
    - Modeliran kao stroj stanja (koncept “trenutne matrice”, “trenutne boje”, …) uz *fixed-function pipeline*
  - v 2.0: 2004., uvođenje podrške za izravan rad sa sjenčarima (vrhova, fragmenata)
    - GLSL (OpenGL Shading Language): jezik za pisanje programa za sjenčare
  - v 4.6: 2017. (trenutno posljednja verzija)

OpenGL / WebGL

- WebGL
  - v 1.0: temeljen na OpenGL ES 2.0, koristi canvas-element u HTML-u, pristupa mu se kroz DOM preko JavaScripta
  - v 2.0: temeljen na OpenGL ES 3.0
  - Ne treba ništa instalirati
    - Programi se pišu kroz JavaScript + GLSL
    - Izvode se izravno u pregledniku
    - Grafika se prikazuje u okviru web-stranice
    - Mogućnostima jednostavniji od punog OpenGL-a

Koordinatni sustavi

- Pri radu s OpenGL-om razlikujemo nekoliko koordinatnih sustava
  - Koordinatni sustav objekta: lokalni koordinatni sustav u kojem je modeliran objekt (npr. kućica)
    - Npr. Ugao kućice je u ishodištu koordinatnog sustava, duljina uzduž osi x je 1, uzduž osi y je 2, visina kućice je 1 u smjeru pozitivne osi z, vrata su u ravnini x=1, krovište je trokutasto visine još 0.5, ukošeno uz dulju stranu kućice

Koordinatni sustavi

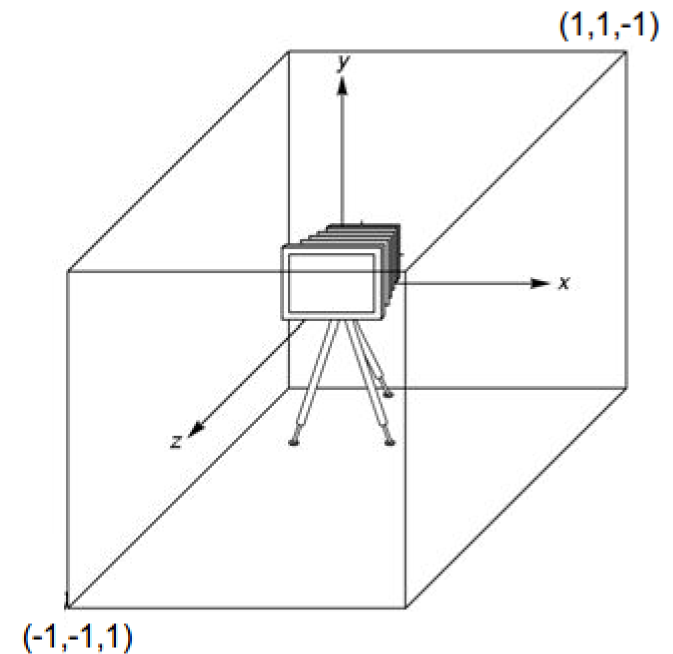
- Pri radu s OpenGL-om razlikujemo nekoliko koordinatnih sustava
  - **Koordinatni sustav scene**: odgovara modeliranom svijetu
    - više primjeraka istog objekta može u scenu biti postavljeno na različita mjesta
    - Npr. Jedna kućica je po y-osi skalirana s 2 i translatirana na lokaciju (0,10,0), druga je zarotirana oko osi z i translatirana na lokaciju (-10,0,0), treća je …
    - **Matrica modela (Model)** koordinate iz sustava objekta preslikava u sustav scene

Koordinatni sustavi

- Pri radu s OpenGL-om razlikujemo nekoliko koordinatnih sustava
  - **Koordinatni sustav kamere**: da bismo dobili prikaz slike, u scenu postavljamo kameru
    - Zanima nas koje koordinate gledano iz koordinatnog sustava kamere imaju objekti
    - Kamera “hvata” jedan dio prostora: volumen pogleda
    - Objekti izvan volumena pogleda se ne prikazuju
    - Preslikavanje iz KS svijeta u KS kamere obavlja **matrica pogleda (View)**

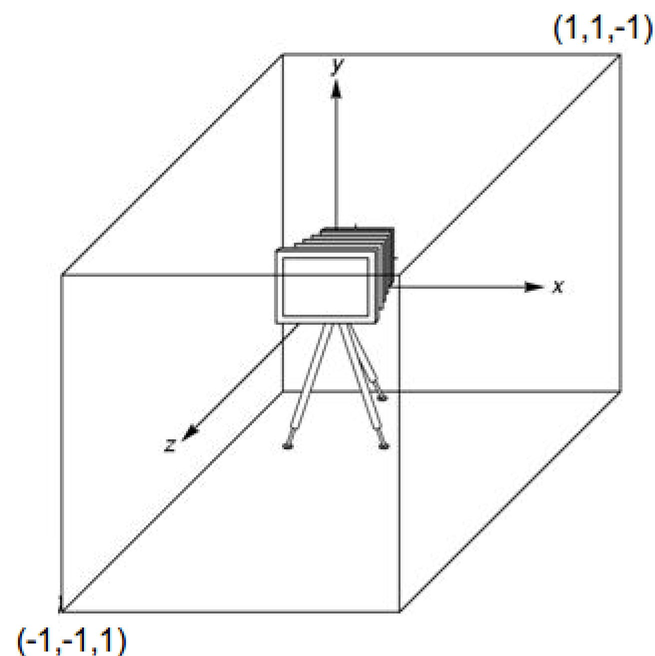
Koordinatni sustavi

- Početno stanje
  - Bez ikakvog podešavanja, kamera je početno postavljena u ishodište koordinatnog sustava scene (0,0,0), “lijevo” gleda uzduž +x-osi, “gore” gleda uzduž +y-osi a kamera je usmjerena prema -z-osi
  - Volumen pogleda je kubus (-1 do +1) uzduž sve tri osi



Početni volumen pogleda

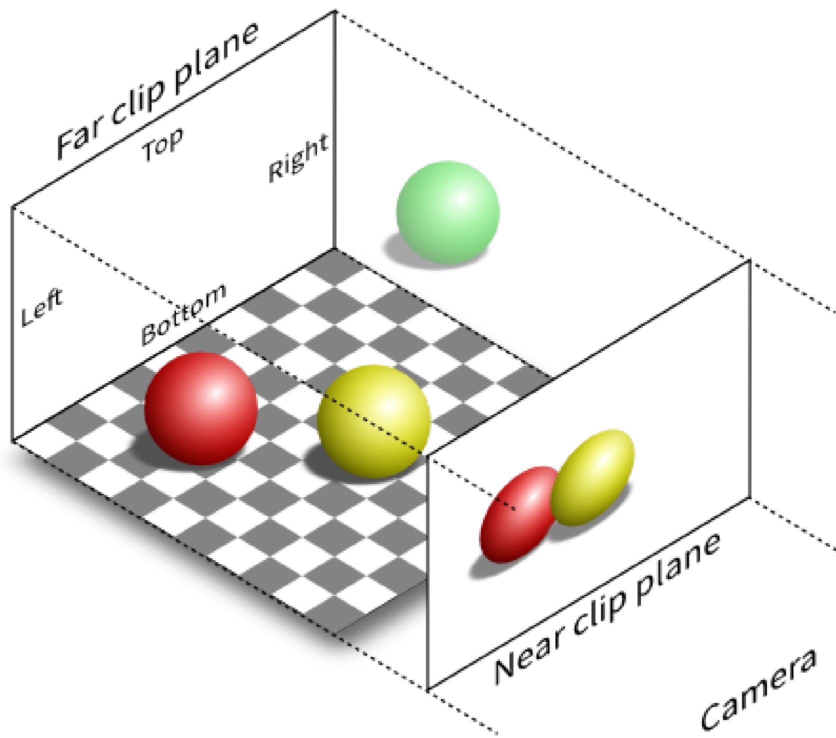
- Početno stanje
  - Točka (0.0,0.0,-0.5) u koordinatnom sustavu svijeta ima koordinate (0.0,0.0,0.5) u početnom koordinatnom sustavu kamere
  - Sve točke koje su izvan ovog volumena pogleda se odsijecaju (engl. *clipping*)



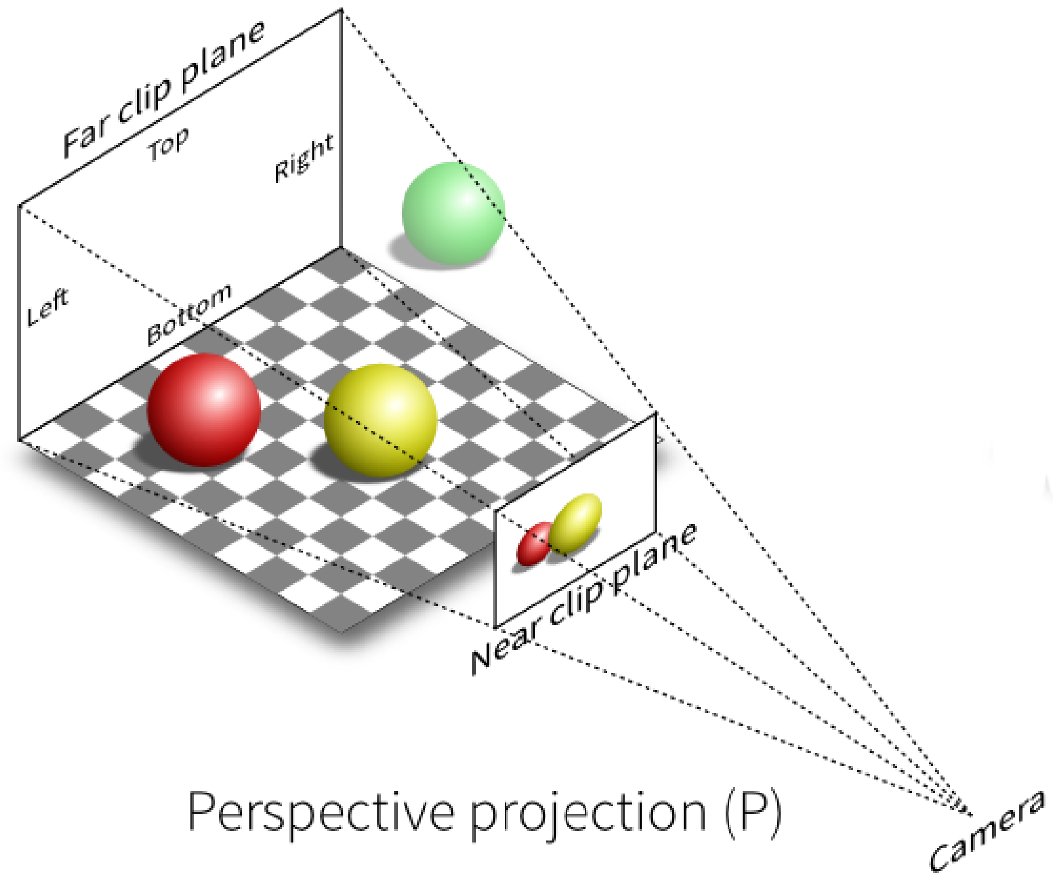
Nastanak slike

- OpenGL sliku stvara tako da točke projicira na ravninu z=-1 (to je “bliža ravnina”) gledano u KS scene uz početne postavke
  - Konceptualno, samo “zaboravimo” z-koordinatu točke iz (x,y,z)
- Potom se ta slika (sjetite se: raspon x i y je -1 do 1) treba pozicionirati na površinu prozora
  - U koje se kvadratno područje to smješta određuje viewport-transformacija

Volumen pogleda / projekcije



Orthographic projection (O)



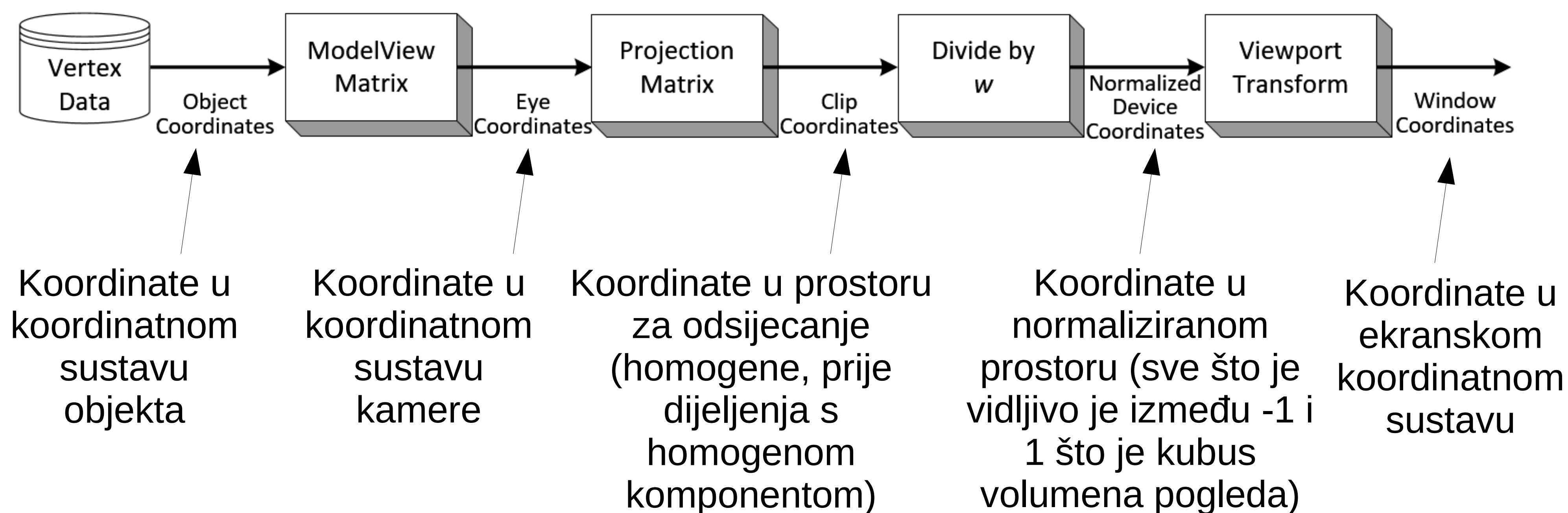
Perspective projection (P)

Početni volumen pogleda (nastavak)

- Volumen pogleda, između ostaloga, je određen bližom ravninom (*near-plane*) i daljom ravninom (*far-plane*)
- Uz početno stanje kamere i smjer gledanja:
  - bliža ravnina udaljena je za 1 od kamere (u KS scene to je ravnina z=-1)
  - dalja ravnina udaljena je za -1 od kamere (u KS scene to je ravnina z=1)
  - Ovime se dogodi "čudna" inverzija (vratit ćemo se na ovo u primjeru 1b)

Operacije: rekapitulacija

- Slika u nastavku prikazuje tok vrhova od objekta do ekrana
  - model*view matrica prikazana je kao jedna transformacija jer je tako OpenGL fiksne strukture vidi



Grafičko korisničko sučelje

- Dio operacijskog sustava zadužen za rad s grafičkim korisničkim sučeljem temeljen je na događajima
  - Kad korisnik pomakne ili klikne mišem: generira se događaj
  - Kada korisnik pritisne ili otpusti tipku na tipkovnici: generira se događaj
  - Kad korisnik promijeni veličinu prozora: generira se događaj
- Događaji se ubacuju u red događaja programa

Grafičko korisničko sučelje

- Programi na najnižoj razini apstrakcije izravno rade s redom događaja
  - `While`-petlja koja vadi opisnik događaja po opisnik događaja i tipično u ogromnoj `if`-`elseif`-`elseif`-... strukturi ili naredbi `switch` obrađuju taj događaj
- Modernije biblioteke omogućavaju rad na višoj razini apstrakcije
  - omogućavaju korisniku da za pojedine vrste događaja registriraju promatrače
  - Sjede u `while`-petlji te pozivaju registriranog promatrača
  - Sjetite se Jave/Swinga: `MouseListener`, ...`Listener`

Grafičko korisničko sučelje

- U primjerima koji slijede koristimo GLUT
  - Omogućava stvaranje i upravljanje prozorom u kojem ćemo crtati grafiku
  - Implementira while-petlju za dostavu događaja (funkcija `glutMainLoop()`)
  - Omogućava registraciju promatrača (u C-u smo pa su to funkcije, predajemo pokazivače na funkcije)
    - `glutReshapeFunc`(pokazivač_na_funkciju)
    - `glutDisplayFunc`(pokazivač_na_funkciju)
    - `glutMouseFunc`(pokazivač_na_funkciju)
    - `glutKeyboardFunc`(pokazivač_na_funkciju)
  - Svaka funkcija mora biti propisane signature: dokumentacija!

Prvi program

- primjer1.c
  - Obratiti pažnju na strukturu programa!
    - `glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)`
    - Metodu `init()`
    - Metodu `myDisplay()`
      - Ne zaboraviti `glFlush()`
    - Metodu `myReshape()`
    - Metodom `glBegin` bira se vrsta primitiva, potom se s `glVertex3f` šalju koordinate vrhova (u trenutku slanja hvata se i veže i trenutna boja za taj vrh), metodom `glEnd` završavamo definiranje objekata

Prvi program

- primjer1.c
  - Bez uporabe dvostrukog spremnika:
    `glutInitDisplayMode(GLUT_SINGLE | ...);`
    metoda `myDisplay()` uobičajeno ima sljedeću organizaciju

```
glClear(…)

// podesi modelview matricu…
// npr. gluLookAt(eye, lookAt, viewUp)

// crtaj
glBegin(…) … glEnd();

// naloži da se buferirane naredbe pošalju na izvođenje
glFlush();
```

Prvi program

- primjer1.c
  - Uz uporabu dvostrukog spremnika:
    `glutInitDisplayMode(GLUT_DOUBLE | ...);`
    metoda `myDisplay()` uobičajeno ima sljedeću organizaciju

```
glClear(…)

// podesi modelview matricu…
// npr. gluLookAt(eye, lookAt, viewUp)

// crtaj
glBegin(…) … glEnd();

// naloži prikazivanje upravo nacrtanog spremnika
glutSwapBuffers();
```

Prvi program: modifikacija

- primjer1b.c
  - Crtamo dva trokuta
    - Što bi trebalo biti prikazano uz početne postavke OpenGL-a?
    - Je li slika u skladu s očekivanjem?

```
glBegin(GL_TRIANGLES);

// Prvi trokut:
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(-0.9f, -0.9f, -0.9f);
glVertex3f( 0.9f, -0.9f, -0.9f);
glVertex3f( 0.0f,  0.9f,  0.9f);

// Drugi trokut:
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3f(-0.9f,  0.9f, -0.9f);
glVertex3f( 0.9f,  0.9f, -0.9f);
glVertex3f( 0.0f, -0.9f,  0.9f);

glEnd();
```

Uporaba z-spremnika

- primjer2.c
  - Možemo tražiti OpenGL da za svaki slikovni element koji “spljošti” **na bližu ravninu**, zapamti koliko je bio udaljen od te ravnine, te da ne dozvoli upisivanje boje tog slikovnog elementa ako je već bio upisan slikovni element koji je bio **bliži**
    - `GlutInitDisplayMode(... | GLUT_DEPTH);`
    - Omogućiti uporabu z-spremnika: `glEnable(GL_DEPTH_TEST);`
    - Prije crtanja nove slike obrisati i z-spremnik: `glClear(... | GL_DEPTH_BUFFER_BIT);`

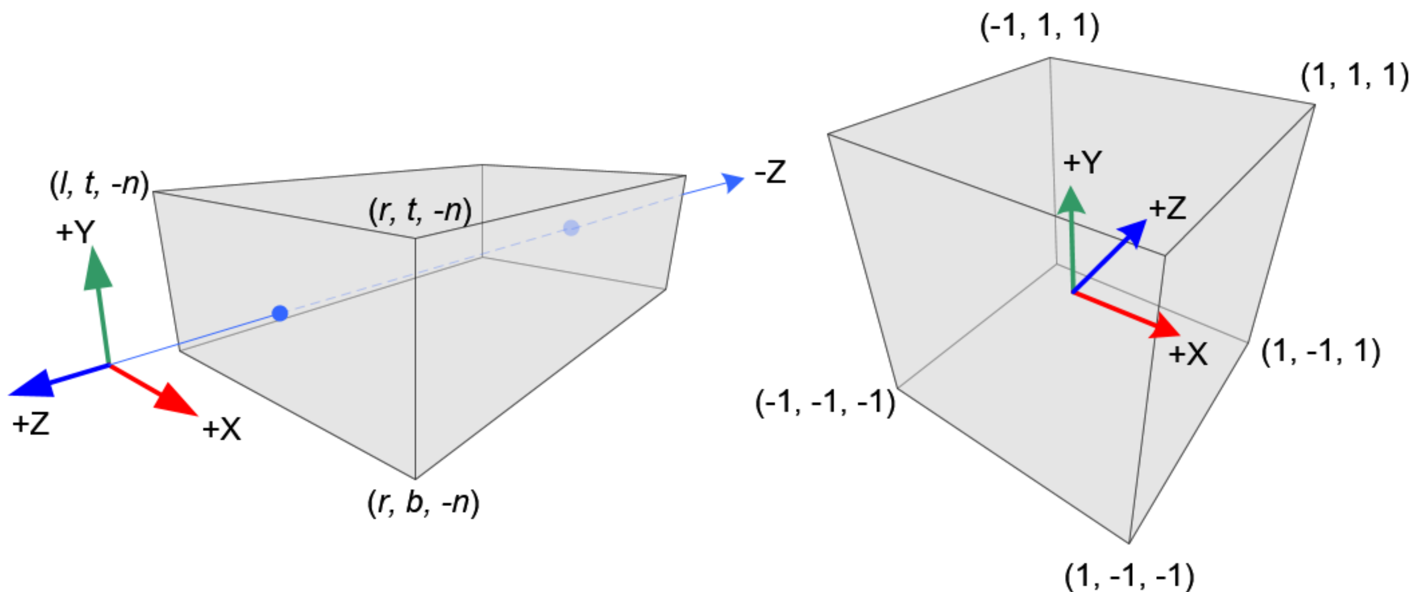
Projekcije

- Ako nam pretpostavljeni volumen pogleda nije odgovarajući, možemo napraviti vlastitu **projekcijsku matricu** koja željeni volumen pogleda transformira u (-1,1) po (x,y,z)
- Uobičajeno:
  - Ortografska projekcija
  - Perspektivna projekcija

Projekcije

- Ortografska projekcija
  - Raspon (l,r) svodi na (-1,1)
  - Raspon (b,t) svodi na (-1,1)
  - Raspon (-n,-f) svodi na (-1,1)
  - `glOrtho(left, right, bottom, top, near, far);`

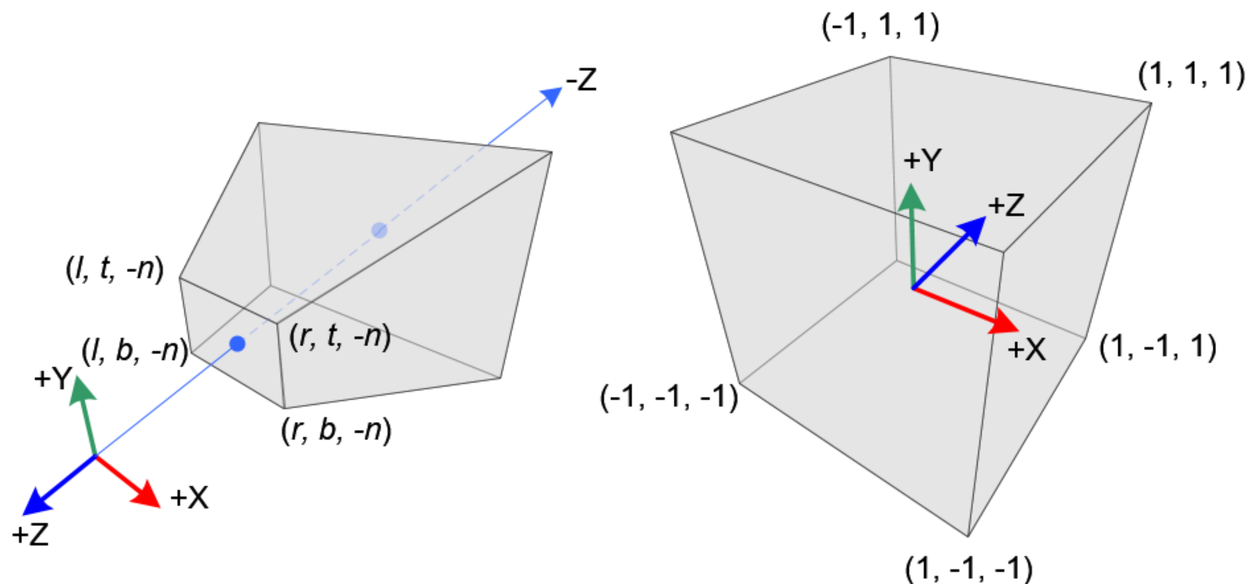
Stvara matricu i trenutnu matricu množi njome te to postavlja kao trenutnu matricu



Projekcije

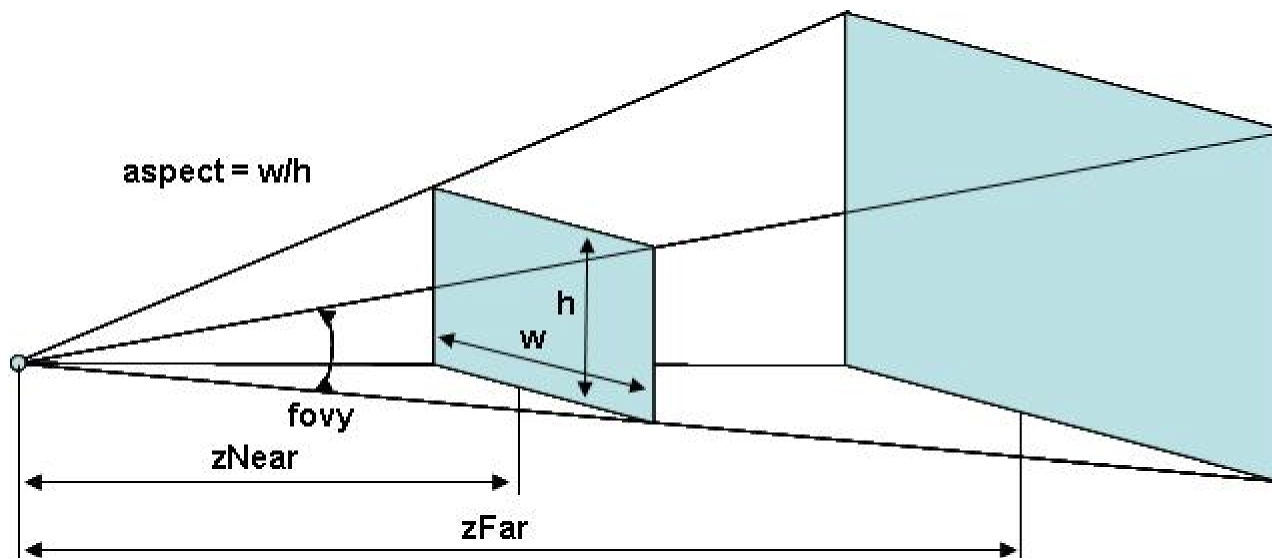
- Perspektivna projekcija
  - Raspon (l,r) svodi na (-1,1)
  - Raspon (b,t) svodi na (-1,1)
  - Raspon (-n,-f) svodi na (-1,1)
  - `glFrustum(left, right, bottom, top, near, far);`

Stvara matricu i trenutnu matricu množi njome te to postavlja kao trenutnu matricu



Projekcije

- Perspektivna projekcija: zgodnija metoda koja na temelju drugog skupa parametara radi isto:
  - Korisnik zadaje kut “vidnog polja” uzduž y-osi (u stupnjevima), omjer širine i visine slike, te udaljenosti ravnina
  - `gluPerspective(fovy, aspect, zNear, zFar);`



Projekcije

- Matrice perspektivne projekcije zadajemo kada smo u modu za podešavanje projekcijske matrice
- Naredba trenutnu projekcijsku matricu množi matricom koju oformi naredba i to postavlja kao novu trenutnu projekcijsku matricu

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0, 1.0, -1.0, 1.0, 1.0, -1.0);
```

Transformacije pogleda

- Obavljaju se u modu za podešavanje matrice *ModelView*
- Naredbe trenutnu MV matricu množe matricom koju oforme naredbe i to postavljaju kao novu trenutnu MV matricu

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslated(2.0, 0.0, 0.0);
GlScaled(1.5,2,1)
```

- Zbog opisanog načina modificiranja MV matrice, posljednje zadana naredba prva djeluje na točke objekta, ...!

Transformacije pogleda

- Preračunavanje koordinata objekata iz sustava scene u sustavu kamere možemo raditi sami
  - Promotrimo početno stanje kamere te objekt čiji je vrh u sustavu scene (3,1,-1)
    - U sustavu kamere njegove su koordinate (3,1,1)
  - Ako kameru pomaknemo udesno (po osi x) na x=1
    - U sustavu kamere vrh će sada imati koordinate (2,1,1)
  - Translacijom kamere udesno (za dx=1) dobili smo isti efekt kao da smo objekte u sceni pomaknuli ulijevo (za dx=-1), bez translacije kamere
  - Primijetite: ta translacija kameru vraća u ishodište!

Transformacije pogleda

- Preračunavanje koordinata objekata iz sustava scene u sustavu kamere možemo raditi sami
  - Promotrimo početno stanje kamere te objekt čiji je vrh u sustavu scene (3,1,-1)
    - U sustavu kamere njegove su koordinate (3,1,1)
  - Ako kameru zarotiramo u smjeru $x \rightarrow z$ za 90°
    - U sustavu kamere vrh će sada imati koordinate (1,1,3)
  - Rotacijom kamere za 90° dobili smo isti efekt kao da smo objekte u sceni zarotirali za -90°
  - Primijetite: ta rotacija kameru vraća u početni smjer gledanja!

Transformacije pogleda

- Stoga, da bismo našli matricu kojom treba pomnožiti koordinate vrhova objekata dane u koordinatnom sustavu scene kako bismo dobili koordinate vrhova objekata u koordinatnom sustavu kamere:
  - Trebamo pronaći matricu koja kameru vraća natrag u početno stanje!

Transformacije pogleda

- Za preračunavanje koordinata u sustavu kamere umjesto da sami računamo matricu, zgodno je osloniti se na metodu gluLookAt(eye, lookAt, viewUp)
  - eye: (x,y,z) koordinate gdje je kamera (očište)
  - lookAt: (x,y,z) koordinate točke u koju je pogled usmjeren (gledište)
  - viewUp: (x,y,z) vektor koji pokazuje "prema gore" za sliku koja će nastati, i on pomaže da se odredi stvarni vektor koji će odgovarati y-osi

Transformacije pogleda

- Za preračunavanje koordinata u sustavu kamere umjesto da sami računamo matricu, zgodno je osloniti se na metodu `gluLookAt(eye, lookAt, viewUp)`
  -

$$\vec{z} = \vec{lookAt} - \vec{eye}$$

$$\vec{x} = \vec{z} \times \vec{viewUp}$$

$$\vec{y} = \vec{x} \times \vec{z}$$

Transformacije pogleda

- OpenGL ne razlikuje matrice modela od matrice pogleda već koristi zajedničku ModelView matricu koja je V*M
- Stoga ako crtamo više objekata, program strukturiramo tako da kao trenutnu matricu postavi matricu pogleda (V) i potom za svaki objekt:
  - Na stog pohrani kopiju trenutne matrice (što je V)
  - Trenutnu matricu modificira (množenjem s M)
  - Pošalje objekt na crtanje (efektivno koristi V*M)
  - Sa stoga restaurira pohranjenu matricu (V)

Transformacije pogleda

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eyex,eyey,eyez, 0.0,0.0,0.0, 0.0,1.0,0.0);
```

Trenutna matrica = V

```
// crtanje objekta 1
glPushMatrix();
glTranslated / glScaled / glRotated
glBegin(...); ... glEnd();
glPopMatrix();
```

<= odgovara matrici M

Trenutna matrica = VM

Trenutna matrica = V

```
// crtanje objekta 2
glPushMatrix();
glTranslated / glScaled / glRotated
glBegin(...); ... glEnd();
glPopMatrix();
```

<= odgovara matrici M

Trenutna matrica = VM

Trenutna matrica = V

Vizualizacija 3D objekata: labosi

- Koristite OpenGL podešen tako da renderira 2D grafiku
  - Matrica modelview: `loadIdentity();`
  - Matrica projekcije: `glOrtho(-w/2,w/2, -h/2, h/2, -1,1)`
  - Viewport-transformacija: `glViewport(0,0,w,h);`
- Možete koristiti `glVertex2d(x,y)` ili čak `glVertex2i(x,y)` za slanje vrhova

Vizualizacija 3D objekata: labosi

- Potom ćete samostalno iz datoteke učitavati objekte te informacije o očištu i gledištu, samostalno odrediti matrice transformacije pogleda i projekcije, svaki vrh njima izmnožiti, podijeliti s homogenom komponentom i dobivene 2D-koordinate izravno slati OpenGL-u na crtanje

Vizualizacija 3D objekata: labosi

- LAB5:
  - Predlažem konstrukciju matrice transformacije pogleda preko viewUp vektora
  - Ravnina projekcije je okomita na pravac očište-gledište
  - Slika nastaje u toj ravnini projekcije
  - Ishodište 2D-koordinatnog sustava podudara se s probodištem pravca očište-gledište i ravnine projekcije
  - Ovisno o dimenzijama učitanog objekta, ili ćete objekt morati naskalirati uniformno, ili projiciranu sliku

Transf. pogleda i projekcija

- primjer3.c
  - Želimo z-spremnik, i želimo gledati na (-1,1)-kubus, ali tako da nam je near-plane na z=1 a far-plane na z=-1 (gledano u KS svijeta)
  - `gluLookAt(0,0,2, 0,0,0, 0,1,0)`
  - `glOrtho(-1,1, -1,1, 1,3)`
- primjer3b.c
  - Idemo trokute pogledati malo ukoso
  - `gluLookAt(2,0,2, 0,0,0, 0,1,0)`
  - `glOrtho(-1,1, -1,1, 1,8)`

Direktno pozivanje myDisplay

- U programima ne želimo direktno pozivati `myDisplay`
- Dobra organizacija programa je sljedeća:
  - Program u podatkovnim strukturama pamti stanje svijeta (gdje se nalazi koji objekt, gdje je kamera i slično)
  - Pod utjecajem korisnika (ili vremena), radi se promjena podataka i poziva `glutPostRedisplay();`
    - Ta će naredba javiti glutu da slika više nije ispravna i da što prije treba ponovno pokrenuti postupak crtanja
    - Glut će dalje pozvati našeg registriranog promatrača

Uporaba z-spremnika

- primjer4.c
  - Mijenjamo položaj kamere kroz vrijeme tako da orbitira oko ishodišta u xz-ravnini i lagano ide gore-dolje
  - Koristimo `gluLookAt`(ex,ey,ez, 0,0,0, 0,1,0) pri čemu kut linearno raste s vremenom (koeficijent *d*) te:

$$e_z = 3 \cdot \cos(d \cdot t)$$

$$e_x = 3 \cdot \sin(d \cdot t)$$

$$e_y = 1 \cdot \sin(8 \cdot d \cdot t)$$

Uporaba sjenčara uz WebGL

- Umjesto cjevovoda fiksirane strukture, omogućava puno fleksibilniji rad
  - Više se ne koristi koncept trenutnih matrica; sami moramo definirati matrice koje ćemo koristiti te napisati program koji ih primjenjuje
  - Vrhovi se šalju u sjenčar vrhova koji ih efektivno priprema za 2D prikaz
  - Potom se radi odsijecanje i po potrebi izračun novih vrhova koji su svi u vidljivom dijelu
  - Radi se viewPort-transformacija čime se dobivaju koordinate vrhova na ekranu
  - Radi se rasterizacija: svaki slikovni element se šalje u sjenčar fragmenata koji za njega mora odrediti konačnu boju

Uporaba sjenčara uz WebGL

- Umjesto cjevovoda fiksirane strukture, omogućava puno fleksibilniji rad
  - Za svaki primitiv koji pošaljemo na crtanje, najprije se vrhovi pošalju u sjenčar vrhova
    - Tamo vrhove možemo transformirati kako god želimo
    - Uobičajeno odradimo množenje PVM * vrh
    - Ovaj sjenčar treba vratiti koordinate vrha u prostoru odsijecanja (to su homogene koordinate koje nakon dijeljenja s homogenom komponentom prelaze u raspon (-1,1) za točke koje su u volumenu pogleda (NDC))
    - Vrhovi koji bi ispali iz volumena pogleda se odsijecaju i rekonstruira se vidljivi dio

Hint: radit ćete algoritme odsijecanja poput Cohen Sutherlanda, Cyrus Becka, ...

Uporaba sjenčara uz WebGL

- Umjesto cjevovoda fiksirane strukture, omogućava puno fleksibilniji rad
  - Jednom kada su utvrđeni/modificirani vrhovi koji su u volumenu pogleda (raspon -1 do 1), koordinate se množe s viewport-transformacijom koja ih preslikava u konačne koordinate slikovnih elemenata na zaslonu (prozoru/slici/…); te su koordinate u ekranskom koordinatnom sustavu

Uporaba sjenčara uz WebGL

- Umjesto cjevovoda fiksirane strukture, omogućava puno fleksibilniji rad
  - Sada, nakon što su utvrđene koordinate vrhova u ekranskom koordinatnom sustavu, radi se rasterizacija
    - Za liniju: računaju se svi slikovni elementi koji čine liniju
    - Za trokut: računaju se svi slikovni elementi koji čine površinu trokuta
    - …
  - Za svaki utvrđeni slikovni element jednom se pokreće sjenčar fragmenata
    - On treba utvrditi boju koja će biti dana tom elementu

Hint: radili ste već Bresenhamov algoritam za linije; implementirali popunjavanje poligona, …

Sjenčar vrhova (vertex-shader)

- Program pisan jezikom GLSL (slično C-u)
- Može specificirati:
  - Nazive *atributa* koje prima za svaki vrh
  - Nazive *varijabli* koje prima "jednom" (vrijednosti ostaju iste su pri obradi svih vrhova; tzv. *Uniform*-i)
    - Uobičajeno koristimo za specificiranje matrica P i V
  - Definira transformirane koordinate vrha zapisivanjem u "posebnu" varijablu `gl_Position`.
  - Može definirati dodatne *izlazne* atribute koje će slati dalje za uporabu u sjenčarima fragmenata

Sjenčar vrhova (vertex-shader)

- Primjer programa koji prima jedan atribut (koordinate vrha) i dva uniforma

```
attribute vec4 aVertexPosition;

uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;

void main() {
  gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
}
```

primijetite tipove podataka koji nam stoje na raspolaganju

Sjenčar fragmenata (fragment-shader)

- Definira boju kojom treba prikazati slikovni element (fragment) zapisivanjem u posebnu varijablu `gl_FragColor` (tipa vec4)
- U najjednostavnijem slučaju, sjenčar fragmenata može svim slikovnim elementima dodijeliti istu boju – primjer u nastavku

```
void main() {
  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);  // RGBA
}
```

Prevođenje programa

- Za uporabu, napisane kodove sjenčara vrhova i fragmenata potrebno je prevesti i polinkati u program koji se zatim može zadati grafičkoj kartici
  - Programski pozivamo prevodilac koji je dio WebGL specifikacije (`shaderCompile`, `linkProgram`)
- Pri prevođenju, svi atributi i uniformi koje smo definirali dobit će redne brojeve ulaza koji su potrebni pri daljnjem crtanju
  - Imamo naredbe `getAttribLocation`(program,ime) i `getUniformLocation`(program,ime)

Specificiranje objekata

- Kad koristimo sjenčare, ne možemo više koristiti `glBegin` / `glEnd` za specificiranje primitiva
- Umjesto toga, koristimo spremnike koji čuvaju sve relevantne informacije
  - Metode: `createBuffer`, `bindBuffer`, `bufferData`
  - Primjerice:
    - sve vrhove spremamo u jedan spremnik
    - sve boje spremamo u drugi spremnik

Crtanje objekata

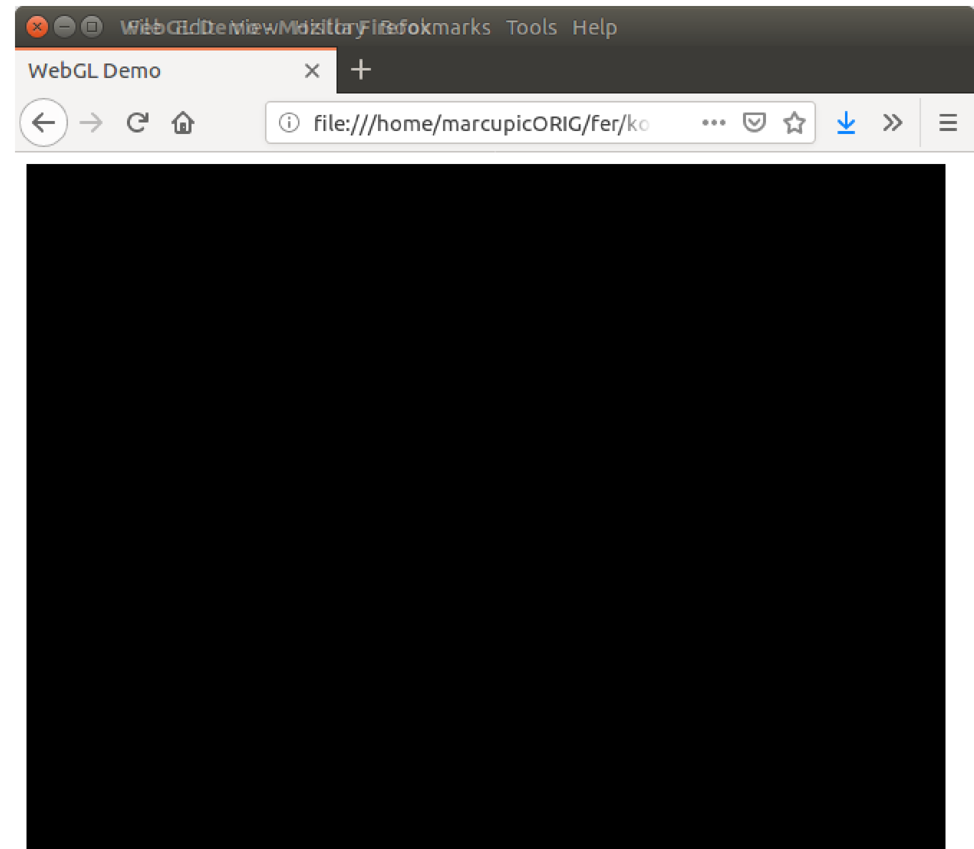
- Da bismo poslali objekt na crtanje, potrebno je:
  - Odabrati koji se prevedeni program koristi za crtanje (`useProgram`)
  - Za svaki od atributa sjenčara vrhova potrebno je omogućiti slanje podataka na taj ulaz (preko rednog broja; `enableVertexAttribArray`) te podesiti iz kojeg spremnika se podatci čitaju i šalju, te kako u spremniku organizirani (`bindBuffer` + `vertexAttribPointer`)
  - Postaviti vrijednosti uniforma (`uniformMatrix4fv`)
  - Pokrenuti obradu primitiva (`drawArrays`)

Primjeri

- JavaScript nema ugrađenu podršku za rad s matricama
  - Treba koristiti neku gotovu biblioteku; koristimo `gl-matrix.js` sa stranice https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial
- primjer1.html: prazno platno
- primjer2.html: naš prvi trokut
- primjer3.html: naša dva trokuta, svaki svojom bojom

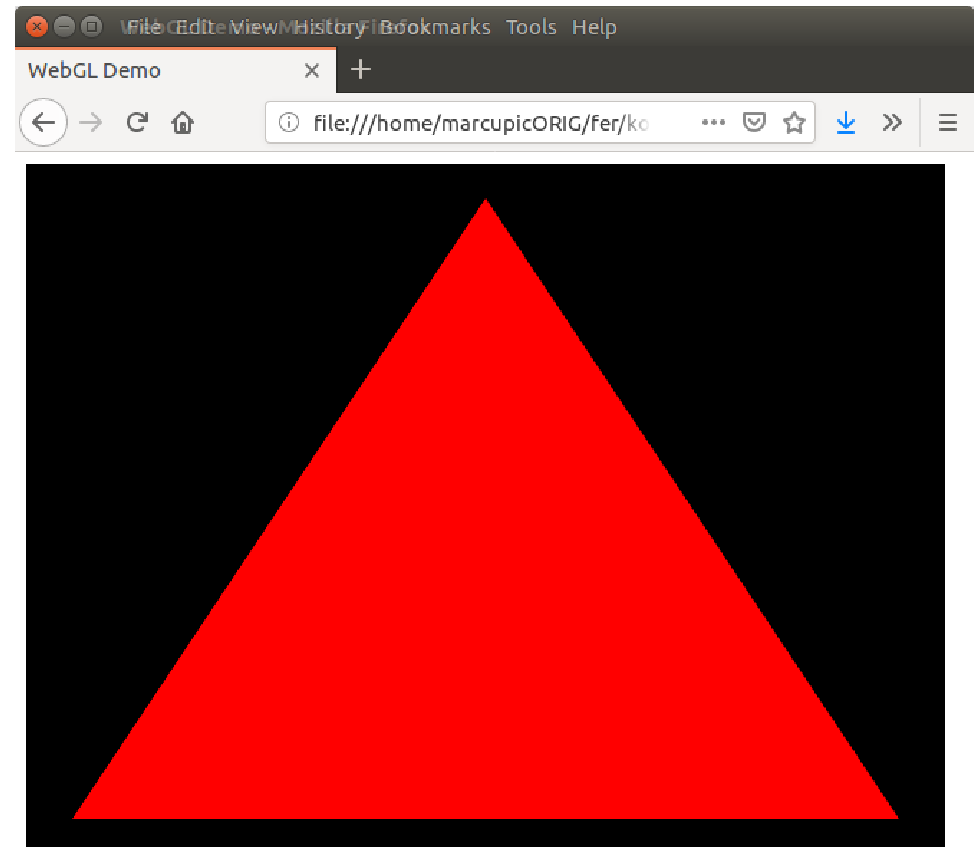
Primjeri

- primjer1.html: prazno platno



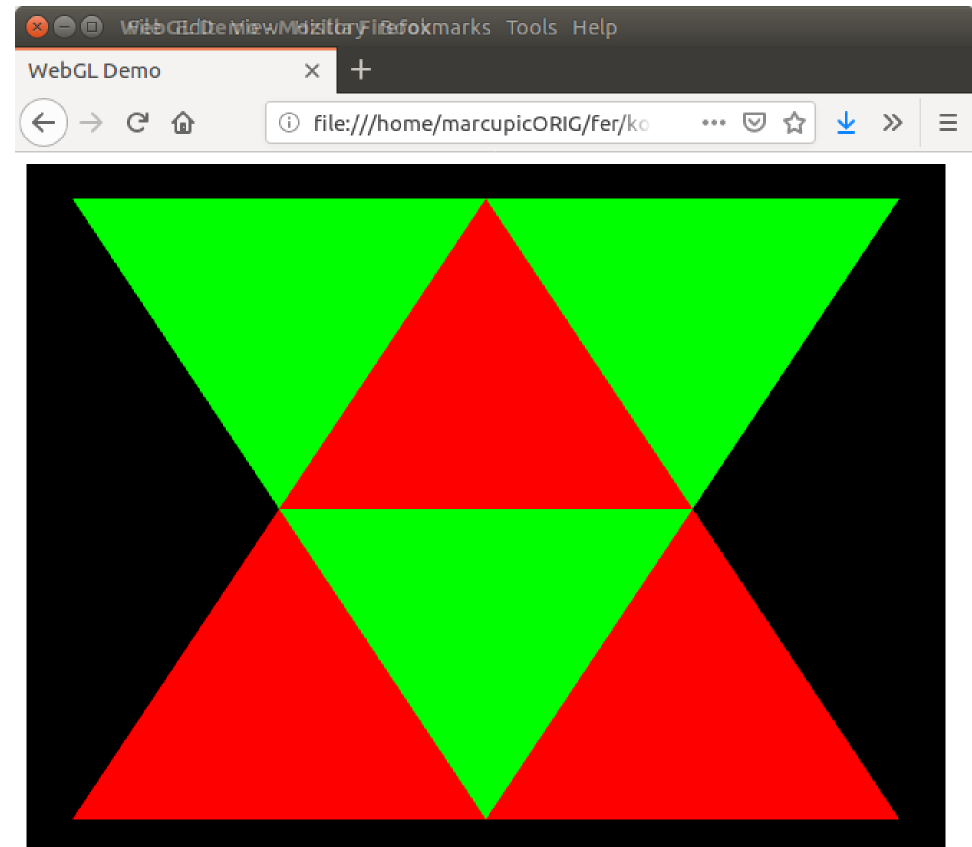
Primjeri

- primjer2.html: naš prvi trokut
  - Jedan spremnik za vrhove koji sadrži 3 vrha



Primjeri

- primjer3.html: naša dva trokuta, svaki svojom bojom
  - Dva spremnika
  - Sjenčar vrhova definira jednu izlaznu varijablu: **`varying lowp vec4 vColor`**
  - Sjenčar fragmenata je definira kao ulaznu; OpenGL interpolira vrijednosti koje se tu pošalju iz vrhova



Primjeri

- Na stranicama IRG-a dostupni su i primjeri koje smo pripremili a koji ilustriraju uporabu sjenčara direktno iz OpenGL-a
  - Programi su pisani u C-u
  - Napravili smo i popratni dokument s uputom
- Primjeri u mnogim minilekcijama iz IRG-a rađeni su izravno u WebGL-u: možete ih pogledati