Računalna grafika: Laboratorijska vježba 3
shader programi i teksture

Toma Sikora[1]

FESB, Sveučilište u Splitu
toma.sikora@fesb.hr

1 Uvod

U trećoj vježbi kolegija Računalna Grafika bavit ćemo se pisanjem shader programa i povezivanjem tekstura na modele učitane kroz datoteke formata .obj. Tokom ove vježbe, koristit ćemo se GLSL jezikom za pisanje shader programa, koje ćemo kao što smo naučili kasnije učitavati, kompajlirati, linkati i izvršavati pomoću OpenGL funkcija. Što se tiče povezivanja tekstura na modele, dodavati ćemo podatke o teksturama u datoteci .obj i povezivati same slike teksture pomoću besplatnog STB Image paketa.

- `https://learnopengl.com/Getting-started/Shaders`
- `https://learnopengl.com/Getting-started/Textures`

Tu možete pronaći dublji opis procesa kojima ćemo se baviti, iako će u materijalima za vježbu biti pripremljen velik dio posla.

2 Shader programi

Kako ste mogli vidjeti na predavanjima, za iscrtavanje 3D objekata i elemenata na 2D zaslonu računala koristimo grafički pipeline, Slika1(`https://learnopengl.com/Getting-started/Hello-Triangle`). To je niz operacija koje na ulazu primaju podatke o tokama u 3D prostoru, obrade ih, te na izlazu proslijede podatke o bojama pojedinih piksela na zaslonu računala.

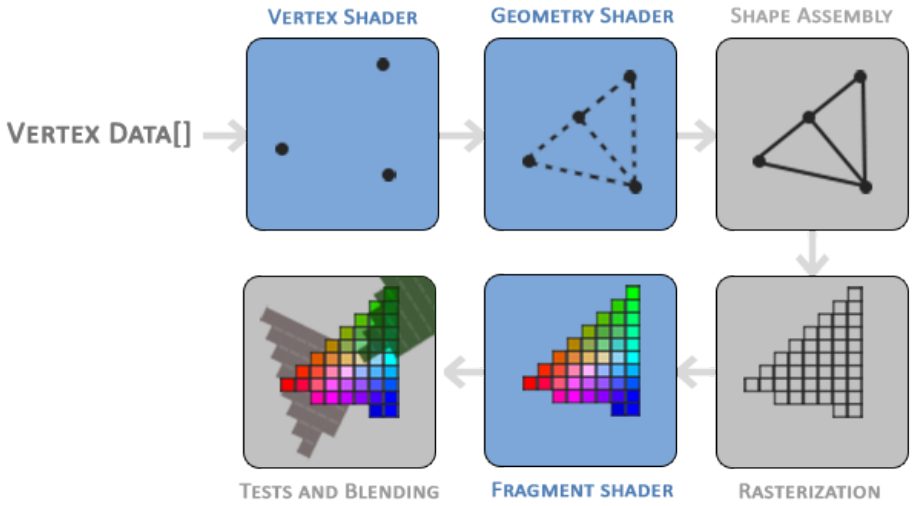

Fig. 1. Grafički *pipeline*

Kao što smo spomenuli u prošloj vježbi, kako bi iscrtali učitane poligone na zaslonu računala, koristimo shader programe. Oni služe za definiciju malih poslova koje treba izvršiti na grafičkoj

kartici i pisani su u GLSL programskom jeziku (u stilu programskog jezika C). Različiti tipovi shader programa izvode se u pojedinim dijelovima grafičkog pipeline-a, npr. vertex shader je prvi korak na ulazu, a fragment shader predzadnji.

2.1 GLSL jezik

Prvi je zadatak u ovoj vježbi upoznavanje sa osnovama GLSL jezika za pisanje shadera. Punim imenom OpenGL Shading Language, jezik je temeljen na C-u te nam omogućuje pravljanje podatacima kao što su boje, vektori i matrice. U njemu pišemo uglavnom kratke i jednostavne programe koji predstavljaju elemente grafičkog pipeline-a. Tipična struktura jednog shader programa izgleda ovako:

```
#version version_number
in type in_variable_name1;
in type in_variable_name2;

out type out_variable_name;

uniform type uniform_name;

void main()
{
    // process input(s) and do some graphics stuff
    ...
    // output processed stuff to output variable
    out_variable_name = stuff_we_processed;
}
```

Svaki GLSL program trebao bi početi sa deklaracijom njegove verzije (npr. `#version 330 core` za OpenGL 3.3). Nakon toga ide lista ulaznih varijabli sa ključnom riječi *in*, lista izlaznih varijabli sa ključnom riječi *out*, te lista globalnih varijabli postavljenih preko CPU sa ključnom riječi *uniform*. Sam kod za izvršavanje stavlja se u main funkciju, u kojoj se procesuiraju spomenute varijable i postavlja vrijednost za izlazne varijable.

2.2 Varijable

Tipovi varijabli koje možemo koristiti su int, float, double, uint i bool, a osim njih GLSL podržava i vektorske i matrične spremnike. Odgovarajući vektorski spremnici su sljedeći:

- *vecn* vektor n varijabli tipa float.
- *bvecn* vektor n varijabli tipa bool.
- *ivecn* vektor n varijabli tipa integer.
- *uvecn* vektor n varijabli tipa unsigned integer.
- *dvecn* vektor n varijabli tipa double.

Mi ćemo uglavnom koristiti *vecn* za rukovanje bojama i 3D koordinatama. Primjer definicije takvih varijabli bi bio: `vec2 vect = vec2(0.5, 0.7);` Što se tiče komunikacije sa ostatkom grafičkog pipeline-a, koristimo sljedeće ključne riječi:

- *in* za ulazne varijable u shader program. Specifično kod vertex shadera, iz razloga što je on na početku grafičkog pipeline-a, prije takvih varijabli definiramo gdje se u buffer-u nalaze njene vrijednosti, npr. `layout (location = 0) in vec3 aPos;` pokazuje da se prve vrijednosti iz buffer-a učitaju u aPos varijablu. S druge strane fragment shader prihvaća vrijednosti varijabli iz prethodnih dijelova grafičkog pipeline-a te kod izgleda ovako: `in vec4 vertexColor;`.

– *out* za izlazne varijable, npr. `out vec4 vertexColor;`.
– *uniform* za varijable koje postavljamo kroz kod, npr. `uniform vec4 specialColor;`. Više o tome kasnije.

2.3 Posebne varijable

Neke su od varijabli unaprijed rezervirane za komunikaciju dijelova grafičkog pipeline-a. To su sljedeće varijable:

– *glPosition* (izlazna varijabla tipa vec4 u vertex shaderu): definira 3D koordinate tog verteksa i postavlja se u main funkciji vertex shadera (ne treba je definirati u listi ulaznih varijabli).
– *FragColor* (izlazna varijabla tipa vec4 fragment shadera): za ispravan fragment shader, trebamo imati jednu izlaznu varijablu kojom ćemo proslijediti vrijednosti boje tog piksela na zaslonu računala. Tu varijablu imenujemo sami, npr. FragColor, te je moramo zapisati u listi izlaznih varijabli.

2.4 Povezivanje varijabli u C/C++ kodu na CPU s varijablama u GLSL na GPU

Kako bismo omogućili dinamičko slanje varijabli u shader programe (postavljali vrijednosti dok se program izvodi, ne hardkodirano) koristimo *uniform* varijable. Nju definiramo slično kao i druge varijable, a vrijednosti postavljamo povezivanjem u glavnom C++ kodu, npr. unutar render petlje. Primjer postavljanja vrijednosti za *uniform* varijablu u render petlji može biti sljedeći:

```
while (!window.isClosed()) {
    glUseProgram(shaderProgram);
    ...
    float timeValue = glfwGetTime();
    float greenValue = (sin(timeValue) / 2.0f) + 0.5f;
    int vertexColorLocation = glGetUniformLocation(shaderProgram, "ourColor");
    glUniform4f(vertexColorLocation, 0.0f, greenValue, 0.0f, 1.0f);
}
```

Pomoću *glGetUniformLocation* prvo dohvatimo indeks lokacije uniform varijable koju smo nazvali "ourColor" u aktivnom shader programu. Nakon toga jednostavno postavljamo vrijednost te varijable pomoću *glUniform4f* funkcije (kraj imena funkcije govori o tipu varijable koji želimo postaviti, npr. 4f su četiri float vrijednosti). Konkretno ovaj kod rezultira u sinusoidnoj promjeni zelene boje vertexa kroz vrijeme.

2.5 Teksture

Do sada smo naučili kako dodavati detalje našem modelu preko boja kojim bojamo svaki od vrhova. Ipak, kada bismo željeli postići veću dozu realizma tom metodom, brzo bismo naišli na probleme. Model bi trebalo nadograditi ogromnim brojem vrhova, te svakom od njih definirati boju kako bi mogli prikazati detalje. Kako bismo preskočili sve te korake, koristimo se teksturama.

One su najčešće 2D slike koje se kasnije projiciraju ili lijepe na poligone u modelu. Na taj način stvaramo iluziju detalja visoke rezolucije bez da moramo povečavati rezoluciju modela.

Kako bismo uputili GPU o načinu na koji treba spojiti dijelove slike teksture sa poligonima modela razvijen je sljedeći standard. Slika se pomoću (u,v) para koordinata, koje mapiraju pravokutnik slike u vrijednostima od 0 do 1 podijeli kao na Slici 2(`https://learnopengl.com/Getting-started/Textures`). Nadalje, za svaki poligon (u našem slučaju trokut), uz svaki vrh, definiramo i (u,v) koordinate točke na teksturi koju vežemo na njega. Ti se podaci zapisuju u vertex buffer-e nakon 3D koordinata vrha. Kod koji je pripremljen za ovu vježbu, te podatke očekuje

u datoteci .obj kao listu linija tipa vt u v. Tih bi linija trebalo biti koliko i vrhova. Preslikavanje (u,v) koordinata sa teksture na model prikazano je na Slici 3(`https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-polygon-mesh/introduction.html`).

Što se tiče učitavanja slike teksture, u kodu pripremljenom za vježbu to se radi pomoću Texture klase na sljedeći način: Texture tex("res/textures/container.jpg");. U suprotnom, ako želite dublje upoznati učitavanje tekstura pogledajte `https://learnopengl.com/Getting-started/Textures`.

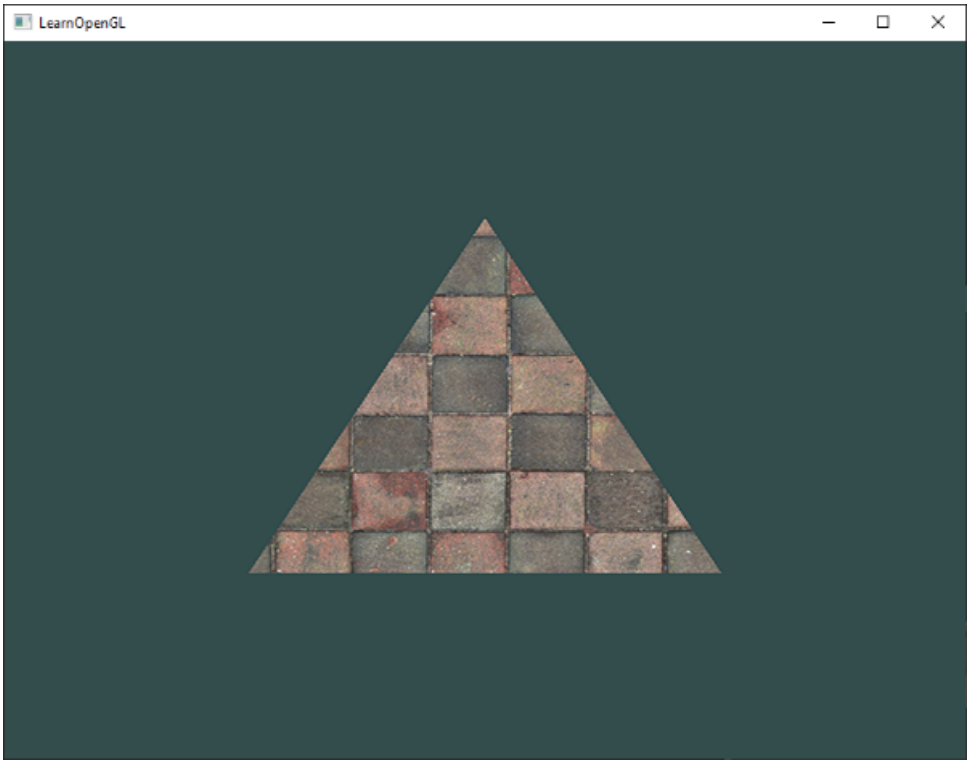

Fig. 2. Mapiranje slike teksture na (u,v) koordinate.

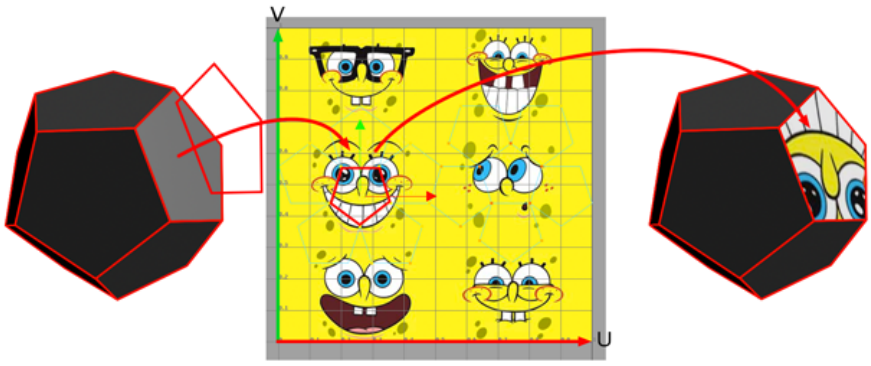

Fig. 3. Preslikavanje (u,v) koordinata.

2.6 Obavezna priprema

Kako ne bi bilo problema sa nadovezivanjem na individualna rješenja iz prošlih vježbi, imat ćete pripremljen kod u kojem je uredno rastavljen i apstraktiran proces učitavanja modela, shadera, tekstura... Kod je dostupan u materijalima vježbi na linku `https://github.com/rperica/GrafikaProjektTemplate/tree/main` i u njemu imate primjer iscrtavanja pravokutnika na koji je zaljepljena jednostavna tekstura. Primjeri shader programa su dani u materijalima vježbe.

Zadaci obavezne pripreme su sljedeći:

1. **Pomicanje modela**: Definirati horizontalni i vertikalni offset u C++ kodu, prenijeti ga preko *uniform* varijable u vertex shader i u shader-u pomaknuti sve vertekse modela za taj offset.
2. **Promjena boje**: Definirati boju preko RGB vrijednosti u C++ kodu (tri float vrijednosti), prenijeti je preko *uniform* varijable u fragment shader i obojati sve vertekse modela u tu boju.
3. **Tekstura**: Uzeti sliku po izboru (primjer container.jpg je dan u materijalima za vježbu) i zalijepiti nju na model pravokutnika.

2.7 Zaključak

U ovoj laboratorijskoj vježbi upoznali smo se s sonovama jezika GLSL i pisanju osnovnih shader programa. Takoder, naučili smo kako teksture učitati i povezati sa poligonima modela koji želimo prikazati. Do sljedećeg puta, koga zanima može provjeriti sljedeće potencijalno korisne linkove:

1. Specifikacija GLSL jezika: `https://www.khronos.org/opengl/wiki/OpenGLShadingLanguage`
2. Opis procesa interpolacije: `https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/interpolation/introduction.html`
3. Uvod u prikazivanje 3D objekta s poligonima: `https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-polygon-mesh/introduction.html`
4. Definiranje materijala putem formata datoteka MTL: `https://paulbourke.net/dataformats/mtl/`