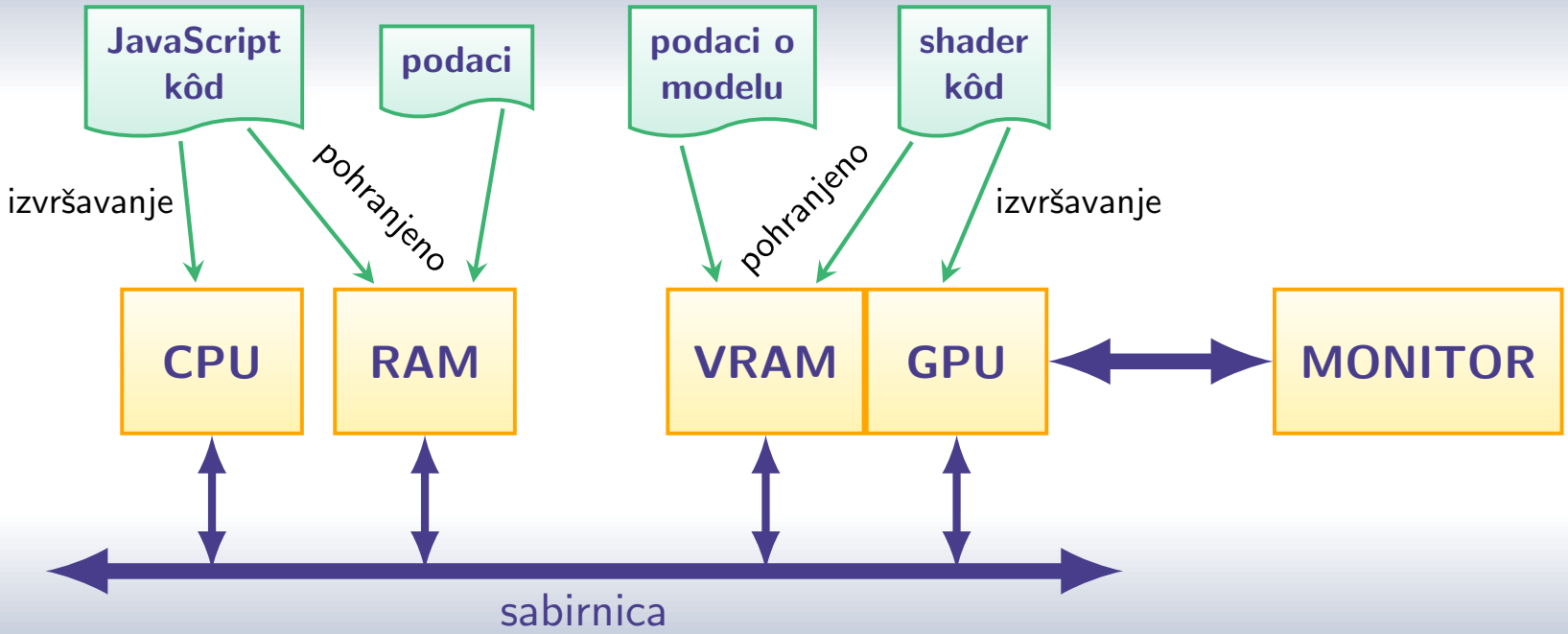
Računalna grafika – laboratorijske vježbe 6

Damir Horvat

Fakultet organizacije i informatike, Varaždin

Sadržaj

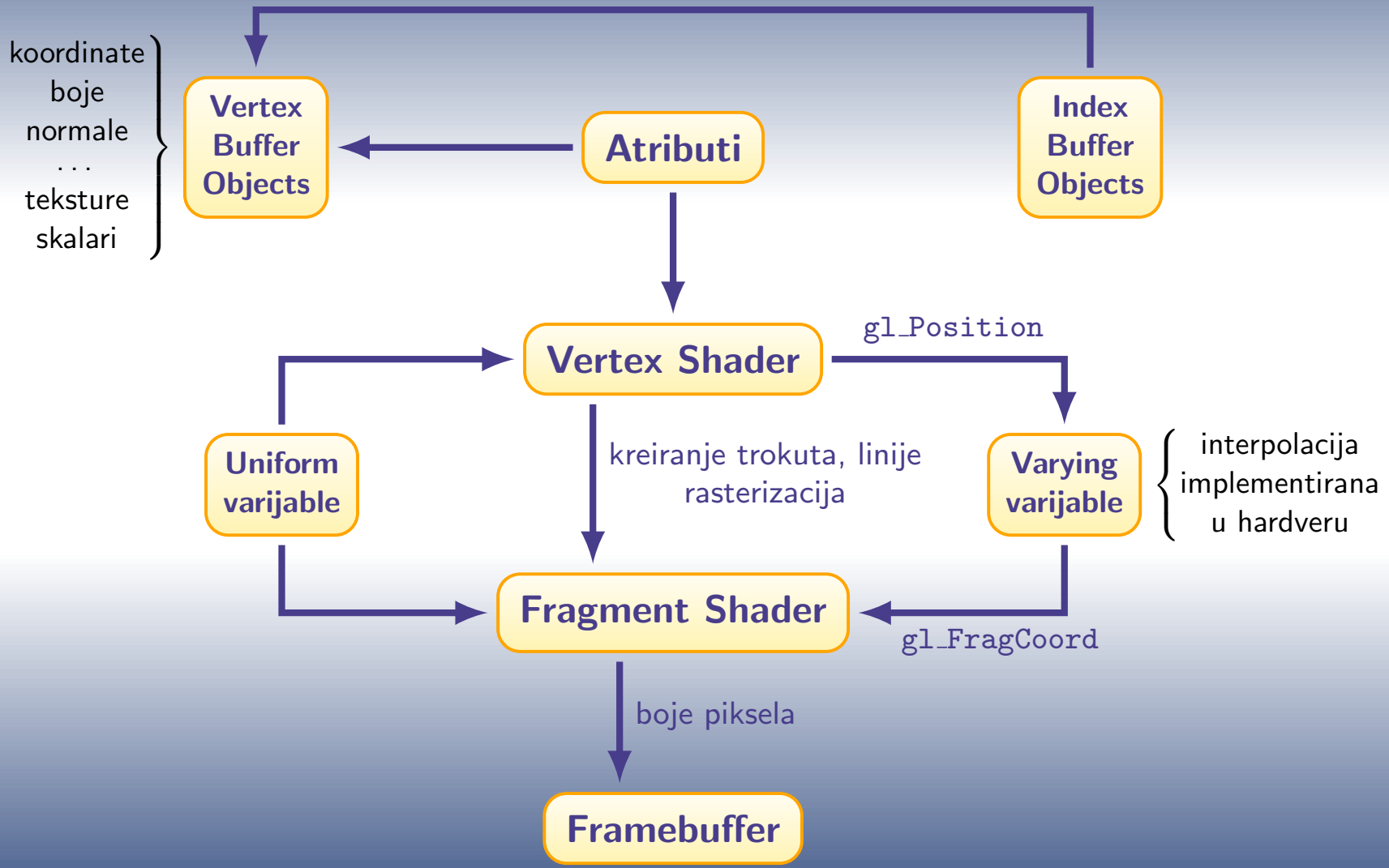
Dijelovi WebGL programa u računalu



Klijent/server model

- JavaScript kôd je klijent.
- GPU je server.
- JavaScript kod šalje naredbe i podatke grafičkoj procesorskoj jedinici (GPU) preko WebGL konteksta.
  - `gl.bufferData(...)`
  - `gl.drawArrays(...)`
  - `gl.clear(...)`
  - ⋮
- WebGL je stroj stanja. Preko WebGL konteksta mijenjamo razna stanja na GPU. Trenutna stanja se koriste kod svakog sljedećeg poziva metoda za iscrtavanje tako dugo dok im ne promijenimo vrijednosti u neke nove koje će onda biti primijenjene na sljedeća iscrtavanja.
- Shader kôd vodi brigu o crtanju objekata na temelju trenutno postavljenih stanja na GPU.

WebGL cjevovod za iscrtavanje



Shader

- Shader se izvodi na grafičkoj kartici (GPU) i zbog toga treba napisati kôd koji će se moći izvoditi na GPU.
- Kôd se daje u obliku dvije funkcije koje se zovu **vertex shader** i **fragment shader** koje se pišu u posebno kreiranom jeziku `GLSL` (GL Shader Language) koji ima sintaksu sličnu `C/C++` programskom jeziku. Zajedno te dvije funkcije zovemo **program**. Više o sintaksi možete vidjeti na ▸ GLSL ES.
- Vertex shader je odgovoran za transformaciju koordinata vrhova objekata u normalizirane koordinate koje se zatim preko `gl.viewport` metode prevode u zaslonske koordinate. Također može računati boju vrha, osvjetljenje i slično ili pak te podatke za pojedini vrh bez dodatnog računanja prosljeđuje preko varying varijabli u fragment shader koji to radi umjesto njega (ovisno o tome kakvi vizualni efekt želimo postići).

Shader

- Vertex shader ne zna ništa o transformacijama koordinata, sve mu moramo sami prirediti i proslijediti preko uniform varijabli. Naše već implementirane klase `MT2D` i `MT3D` možemo iskoristit u tu svrhu ili pak uzeti neku gotovu biblioteku koja zna raditi s transformacijama, npr. ▸ glMatrix. `GLSL` jezik ima ugrađene sve potrebne matematičke operacije za rad s vektorima i matricama.
- Na ovim vježbama radimo samo s normaliziranim koordinatama pa danas nećemo trebati `MT2D` i `MT3D` klase. Normalizirane koordinate su uvijek između -1 i 1 i jedino s njima WebGL zna raditi, tj. prevesti ih u zaslonske koordinate preko `gl.viewport` metode.

Shader

- Svaki osnovni objekt (točka, linija, trokut) mora se rasterizirati prije prikaza na zaslonu. Rasterizacija je već implementirana u hardveru, a fragment shader se brine za boju svakog pojedinog piksela od kojih se rasterizirani objekt sastoji. Fragment shader uz uniform varijable koristi sve proslijeđene podatke za pojedini vrh iz vertex shadera kako bi odredio boju vrha, tj. boju piksela u koji će se taj vrh preslikati. Boje svih ostalih piksela koji ne pripadaju vrhovima rasteriziranog objekta određuje interpolacijom dostupnih podataka od vrhova. Proces interpolacije je također implementiran u hardveru, a kôd od fragment shadera samo daje upute kako odrediti boju pojedinog fragmenta (piksela).

Shader

- Na primjer, trokut se sastoji od tri vrha. Međutim, osim ta tri vrha, treba obojati i unutrašnjost tog trokuta (ukoliko želimo obojani trokut). Boju pojedinog vrha trokuta možemo već odrediti i u vertex shaderu ili pak to možemo prepustiti fragment shaderu (ovisno kakav vizualni efekt želimo postići). Sve ostale fragmente trokuta tek možemo obojati nakon rasterizacije trokuta, tj. u fragment shaderu preko već spomenute automatizirane interpolacije na temelju dostupnih podataka o vrhovima.
- Fragment osim informacije o boji može sadržavati i neke druge informacije: koordinate normale, z-koordinatu za testiranje vidljivosti i mnoge druge podatke. Za piksel na ekranu je bitna samo konačna boja, dok fragment sadrži više informacija. Dakle, prikazani fragment na ekranu je zapravo piksel. Fragment osim boje sadrži općenito i neke druge informacije koje se mogu dobiti preko ugrađenih naredbi u `GLSL` jeziku, dok piksel sadrži samo informaciju o boji.

Framebuffer

- Framebuffer je dvodimenzionalni buffer u kojemu su spremljeni svi procesirani fragmenti. Nakon što su obrađeni svi fragmenti od svih objekata koje želimo prikazati na ekranu, kreira se 2D slika i prikazuje na zaslonu. Osim određivanja boje, u 3D sceni treba testirati i vidljivost pomoću spremnika dubine (z-buffer) i slično (ovisno o tome kakve se kompleskne zahvate radimo). Slikovito rečeno, framebuffer se sastoji od fragmenata, a zaslon se sastoji od piksela.

VBO i IBO

- Vertex buffer object (VBO) sadrži podatke o geometriji objekta koji se želi nacrtati: koordinate vrhova, normale vrhova, boje vrhova, koordinate teksture za pojedini vrh, . . .
- Index buffer object (IBO) sadrži informacije o odnosima vrhova u VBO. Prilikom iscrtavanja objekata koristi se indeks svakog vrha iz VBO kao vrijednost. Na primjer, ako prilikom iscrtavanja objekta koristimo isti vrh više puta, kako bismo uštedili memoriju i smanjili broj poziva vertex shader programa, ne unosimo ga u VBO više puta, nego radije koristimo isti indeks iz IBO više puta. Indeksi govore WebGL-u kako povezati vrhove iz VBO u cjelinu.
- U nekim situacijama možda moramo u VBO unositi isti vrh više puta (npr. ako isti vrh ima više različitih normala i slično). Zapravo sve ovisi o situaciji i vizualnom efektu koji želimo postići.

Atributi

- Ulazne varijable za vertex shader koje opisuju kako izvući podatke iz buffera.
- Na primjer, pretpostavimo da su u nekom bufferu spremljene 3D koordinate vrhova kao uređene trojke 32-bitnih realnih brojeva. Atribut je varijabla koja mora znati iz kojeg buffera treba izvući podatke i kojeg su tipa podaci (u našem slučaju se radi o uređenim trojkama 32-bitnih realnih brojeva), s kojim pomakom u bajtima (offset) od početne pozicije treba startati (možda ne želimo izvući sve podatke iz buffera), jesu li koordinate vrhova isprepletene s nekim drugim podacima u tom bufferu (bojama vrhova, normala i slično) i koliko su daleko u bajtima podaci o koordinatama dva uzastopna vrha (stride).
- S obzirom da se vertex shader poziva na svakom vrhu, atributi će imati svaki put drukčije vrijednosti kada je vertex shader pozvan.

Uniform varijable

- Ulazne varijable koje su dostupne i vertex shaderu i fragment shaderu.
- Za razliku od atributa, uniform varijable su konstantne tijekom jednog pozvanog ciklusa iscrtavanja.
- Na primjer: pozicija svjetla i njegova svojstva, svojstva kamere, matrice transformacija se modeliraju kao uniform varijable.
- Uniform varijable su zapravo globalne varijable koje postavljamo prije pozivanja shader programa.
- U WebGL 2.0 verziji uniform varijable također mogu biti spremljene u memoriji grafičke kartice (uniform buffers). O tome će biti više riječi kasnije kod modela osvjetljavanja.

Varying varijable

- Koriste se za prenošenje podataka iz vertex shadera u fragment shader.
- Ovisno o tome što se iscrtava (točke, linije, trokuti), vrijednosti varying varijabli iz vertex shadera bit će interpolirane prilikom izvođenja fragment shadera.
- U WebGL 1.0 verziji, varying varijable su se deklarirale s rezerviranom riječi `varying` u oba shadera.
- U WebGL 2.0 verziji, varying varijable u vertex shaderu se deklariraju s rezerviranom riječi `out`, a u fragment shaderu s rezerviranom riječi `in`.

Kompajliranje shadera

```
function makeShader(gl,source,type) {
 //napravi shader objekt
 var shader = gl.createShader(type);
 //pridruzi shader source kod
 gl.shaderSource(shader,source);
 //kompajliraj shader
 gl.compileShader(shader);
 //provjeri je li sve ok
 var uspjeh=gl.getShaderParameter(shader,gl.COMPILE_STATUS);
 if (!uspjeh) {
   throw "Shader nije kompajliran: " + gl.getShaderInfoLog(
    shader);
 }
 return shader;
}
```

- gl.createShader
- gl.shaderSource
- gl.compileShader
- gl.getShaderParameter
- gl.getShaderInfoLog

Povezivanje shadera u program

```
function makeProgram(gl,vshader,fshader) {
 var program = gl.createProgram();
 //pridruzi shadere
 gl.attachShader(program,vshader);
 gl.attachShader(program,fshader);
 //povezi shadere u program
 gl.linkProgram(program);
 //provjeri je li dobro povezano
 var uspjeh=gl.getProgramParameter(program,gl.LINK_STATUS);
 if (!uspjeh) {
   throw "Program nije kreiran kako treba: "
         + gl.getProgramInfoLog(program);
 }
 return program;
}
```

- gl.createProgram
- gl.attachShader
- gl.linkProgram
- gl.getProgramParameter
- gl.getProgramInfoLog

Kompajliranje shadera iz <script> oznake

```
function kompajlirajShader(gl,id,type) {
  var shaderSkripta = document.getElementById(id);
  if (!shaderSkripta) {
    throw "Nepoznata skripta: " + id;
  }
  var shaderSource = shaderSkripta.text.trim();
  if (!type) {
    if (shaderSkripta.type == "x-shader/x-vertex") {
      type = gl.VERTEX_SHADER;
    } else if (shaderSkripta.type == "x-shader/x-fragment") {
      type = gl.FRAGMENT_SHADER;
    } else if (!type) {
      throw "Nije specificiran tip shadera."
    }
  }
  return makeShader(gl,shaderSource,type);
}
```

Napravi program iz <script> oznake

```
function napraviProgram(gl,vsID,fsID) {
  var vertexShader = kompajlirajShader(gl,vsID);
  var fragmentShader = kompajlirajShader(gl,fsID);
  return makeProgram(gl,vertexShader,fragmentShader);
}
```

Struktura WebGL programa

1. Kreirati canvas element i dobiti WebGL kontekst.
2. Inicijalizirati prozor za prikaz (viewport).
3. Kreirati jedan ili više buffera koji sadrže podatke za iscrtavanje (koordinate vrhova, boje vrhova, normale, koordinate teksture, skalare,...).
4. Kreirati matrice za transformacije vrhova spremljenih u bufferima za prikaz objekata u canvasu (geometrijske transformacije, transformacije pogleda, projiciranje).
5. Napisati jedan ili više shadera u kojima se implemetira algoritam za iscrtavanje 3D (ili 2D) scene na canvasu.
6. Povezati shadere s parametrima: povezati atribute s kreiranim bufferima, postaviti `uniform` varijable, proslijediti podatke iz vertex shadera u fragment shader preko `varying` varijabli.
7. Nacrtati sliku pomoću WebGL metoda za crtanje: `drawArrays` i/ili `drawElements`.

Kreiranje buffera VBO

- Kreiranje geometrije preko JavaScript polja

```
//vrhovi kvadrata u ravnini
var vrhovi = [-0.5, -0.5, //lijevi donji vrh
               0.5, -0.5, //desni donji vrh
               0.5, 0.5,  //gornji desni vrh
               -0.5, 0.5]; //gornji lijevi vrh
```

- Kreiranje buffer objekta ▸ gl.createBuffer

```
var vrhoviBuffer = gl.createBuffer();
```

- Postavljanje tekućeg buffera u kojeg će se prenijeti podaci. Prvi parametar je vrsta buffera, a drugi ime buffer objekta. ▸ gl.bindBuffer

```
gl.bindBuffer(gl.ARRAY_BUFFER, vrhoviBuffer);
```

Kreiranje buffera VBO

- Prenošenje podataka iz radne memorije u memoriju grafičke kartice u trenutno aktivni buffer ▸ gl.bufferData

```
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vrhovi),
    gl.STATIC_DRAW);
```

  - Prvi parametar je vrsta buffera.
  - Drugi parametar su podaci (tipizirano JavaScript polje). Više o tome možete pogledati na ▸ JavaScript typed arrays. Nama treba samo `Float32Array` za VBO te `Uint8Array` i `Uint16Array` za IBO. Uočite koliko se bajtova koristiti za svaki element koji je spremljen u takvim poljima.
  - Treći parametar je savjet WebGL-u za efikasniji rad s bufferom. Mi ćemo uglavnom koristiti `gl.STATIC_DRAW`.

Kreiranje buffera VBO

- `gl.STATIC_DRAW`: podaci u bufferu se neće mijenjati i koristit će se mnogo puta za iscrtavanje
- `gl.DYNAMIC_DRAW`: podaci u bufferu će se puno puta mijenjati i koristit će se puno puta za iscrtavanje
- `gl.STREAM_DRAW`: podaci u bufferu se neće mijenjati i koristit će se nekoliko puta za iscrtavanje

`WebGL2` kontekst ima još nekoliko dodatnih takvih savjeta koje možete pogledati na ▸ gl.bufferData.

Kreiranje buffera VBO

- Konačno, dobra je praksa "osloboditi" tekući VBO buffer kako ga zabunom ne bismo mijenjali ili koristili.

```
gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

Kreiranje buffera IBO

- Kreiranje indeksa preko JavaScript polja

```
var indeksi = [0, 1, 2, 0, 2, 3];
```

- Kreiranje buffer objekta ▸ gl.createBuffer

```
var indeksiBuffer = gl.createBuffer();
```

- Postavljanje tekućeg buffera u kojeg će se prenijeti podaci. Prvi parametar je vrsta buffera, a drugi ime buffer objekta. ▸ gl.bindBuffer

```
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indeksiBuffer);
```

Kreiranje buffera IBO

- Prenošenje podataka iz radne memorije u memoriju grafičke kartice u trenutno aktivni buffer ▸ gl.bufferData

```
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint8Array(
    indeksi), gl.STATIC_DRAW);
```

  - `Uint8Array` koristimo ako su svi indeksi između 0 i 255. Za svaki indeks u bufferu koristi se jedan bajt.
  - `Uint16Array` koristimo ako postoje indeksi strogo veći od 255. Najveći dozvoljeni indeks u tom slučaju je $65\,535$. Za svaki indeks u bufferu se koriste dva bajta.
  - `WebGL` je trenutno ograničen samo na 16-bitne indekse. Ukoliko je za neki objekt potrebno indeksiranje s indeksima većima od $65\,535$, njegovo iscrtavanje se mora obaviti preko nekoliko poziva naredbi za crtanje.

Kreiranje buffera IBO

- Konačno, dobra je praksa "osloboditi" tekući IBO buffer kako ga zabunom ne bismo mijenjali ili koristili.

```
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, null);
```

Povezivanje atributa s VBO bufferima

Svaki atribut u vertex shaderu je povezan s jednim i samo jednim VBO bufferom. Može više različitih atributa pokazivati na isti buffer ili neke njegove dijelove. Pretpostavimo da smo napravili program pomoću naše naredbe `napraviProgram` i da smo dobili lokaciju atributne varijable `a_position` u shader programu. ▸ gl.getAttribLocation

```
program = napraviProgram(gl, "vertex-shader", "fragment-
    shader");
//koji program koristiti (mozemo ih imati vise)
gl.useProgram(program);
//dobivanje lokacije atributa a_position
program.a_position = gl.getAttribLocation(program, "
    a_position");
```

Povezivanje atributa s VBO bufferima

Želimo povezati atribut `a_position` s ranije kreiranim bufferom `vrhoviBuffer`.

- Najprije treba postaviti tekući buffer.

```
gl.bindBuffer(gl.ARRAY_BUFFER, vrhoviBuffer);
```

- Sljedeći korak je povezivanje atributa (preko njegove dobivene lokacije) s tekućim bufferom. ▸ gl.vertexAttribPointer

```
gl.vertexAttribPointer(program.a_position, 2, gl.FLOAT,
    false, 0, 0);
```

Povezivanje atributa s VBO bufferima

```
gl.vertexAttribPointer(index, size, type, normalized, stride,
    offset);
```

- `index`: lokacija atributa kojeg želimo modificirati
- `size`: koliko podataka iz buffera uzimamo za pojedini vrh. U našem slučaju su to dva podatka (x i y koordinata vrha). Općenito je `size` jednak $1, 2, 3$ ili 4.
- `type`: tip podataka u bufferu. U našem slučaju su u bufferu realni brojevi pa je tip jednak `gl.FLOAT`. Mogući su još neki drugi tipovi podataka, za detalje pogledajte ▸ gl.vertexAttribPointer.
- `normalized`: treba li podatke u bufferu normalizirati (`true`) ili ih ostaviti onakvima kakvi jesu (`false`).
- `stride`: jesu li podaci u bufferu jedan za drugim (`stride=0`) ili se radi o isprepletenom bufferu s nekim drugim podacima. U tom slučaju `stride` daje udaljenost u bajtima od početnih pozicija između dva uzastopna istovrsna podatka.
- `offset`: pomak u bajtima od početka buffera od kuda treba početi uzimati podatke. Mora biti višekratnik od broja bajta koji se koriste za `type`.

Povezivanje atributa s VBO bufferima

- Također, treba omogućiti korištenje pojedinog vertex shader atributa preko njegove dobivene lokacije. ▸ gl.enableVertexAttribArray

```
gl.enableVertexAttribArray(program.a_position);
```

- Konačno, dobra je praksa "osloboditi" tekući VBO buffer kako ga zabunom ne bismo mijenjali ili koristili.

```
gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

Isprepleteni buffer

- U istom bufferu su spremljene 2D koordinate četiri vrha kvadrata i boje za pojedini vrh koje su zadane preko RGB komponenti.
- Pretpostavimo da su u vertex shaderu dva atributa `a_position` i `a_color` za 2D koordinate vrhova kvadrata i njihove boje.
- Pretpostavimo smo napravili shader program spremljen u varijablu `program` i da su lokacije spomenutih atributa spremljene u `program.a_position` i `program.a_color`.
- Pogledajmo kako te atribute povezati s isprepletenim bufferom. Tu su bitni ranije spomenuti parametri `stride` i `offset`.

Isprepleteni buffer

```
var vrhovi = [
    -0.5,-0.5, 1,0,0, //lijevi donji vrh, crvena boja
    0.5,-0.5, 0,1,0, //desni donji vrh, zelena boja
    0.5,0.5, 0,0,1,  //gornji desni vrh, plava boja
    -0.5,0.5, 1,1,0]; //gornji lijevi vrh,zuta boja
  ...

gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vrhovi), gl.
    STATIC_DRAW);

gl.vertexAttribPointer(program.a_position, 2, gl.FLOAT, false
    , 20, 0); //5*4=20 bajta (stride)

gl.vertexAttribPointer(program.a_color, 3, gl.FLOAT, false,
    20, 8); //5*4=20 bajta (stride), 2*4=8 bajta (offset)
  ...
```

Isprepleteni buffer

```
gl.vertexAttribPointer(program.a_position, 2, gl.FLOAT, false
    , 20, 0);
```

- Za koordinate svakog vrha kvadrata se koriste dva podatka.
- Koordinate prvog vrha kvadrata su napisane odmah na početku buffera pa je `offset=0`.
- Nakon koordinata prvog vrha slijede tri podatka za boju prvog vrha koje treba preskočiti kako bismo došli do koordinata drugog vrha. Međutim, gleda se koliko treba podataka preskočiti od početne pozicije koordinata prvog vrha do početne pozicije koordinata drugog vrha. U našem slučaju je to 5 podataka, a za svaki podatak se koriste u bufferu 4 bajta (unutra su realni brojevi) pa je razmak između koordinata svakog vrha jednak 20 bajta. Stoga je `stride=20`.

$$\underbrace{-0.5, -0.5,\ 1, 0, 0,}_{\text{(5 podataka)}\cdot\text{(4 bajta)}}\ 0.5, -0.5,\ 0, 1, 0,\ 0.5, 0.5,\ 0, 0, 1,\ -0.5, 0.5,\ 1, 1, 0$$

Isprepleteni buffer

```
gl.vertexAttribPointer(program.a_color, 3, gl.FLOAT, false,
    20, 8);
```

- Za boju svakog vrha kvadrata se koriste tri podatka.
- Podaci za boju prvog vrha napisani su nakon koordinata prvog vrha pa treba početi čitati buffer nakon prva dva podatka, tj. nakon 8 bajta od početka buffera. Stoga je `offset=8`.
- Kako bismo došli do podataka za boju drugog vrha, treba preskočiti dva podatka za koordinate drugog vrha. Međutim, gleda se koliko treba podataka preskočiti od početne pozicije boje prvog vrha do početne pozicije boje drugog vrha. U našem slučaju je to 5 podataka, a za svaki podatak se koriste u bufferu 4 bajta (unutra su realni brojevi) pa je razmak između boja svakog vrha jednak 20 bajta. Stoga je `stride=20`.

$$\underbrace{-0.5, -0.5,}_{\text{(2 podatka)}\cdot\text{(4 bajta)}}\ \underbrace{1,0,0,\ 0.5, -0.5,}_{\text{(5 podataka)}\cdot\text{(4 bajta)}}\ 0,1,0,\ 0.5,0.5,\ 0,0,1,\ -0.5,0.5,\ 1,1,0$$

VAO objekti

- **Vertex array objects** (VAOs) omogućuju lokalno spremanje informacija o atributima i bufferima u objekt s kojim je lako upravljati.
- U VAO objekte se spremaju sljedeće informacije: stanja atributa, koji buffer koristiti za pojedini atribut i kako preuzeti podatke iz buffera.
- Bitno je koristiti VAO objekte jer oni značajno skraćuju vrijeme potrebno za iscrtavanje. Bez VAO objekata sve informacije o atributima su u globalnom `WebGL` stanju, što znači da pozivanjem metoda `gl.vertexAttribPointer`, `gl.enableVertexAttribArray`, `gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, ime)` utječemo na globalno stanje `WebGL` konteksta. To dovodi do usporavanja iscrtavanja jer prije svakog crtanja trebamo postavljati atribute i buffere. VAO objekti omogućuju da to izbjegnemo.

VAO objekti

- Kreiranje VAO objekta ▸ gl.createVertexArray

```
var vao;
 ...
vao = gl.createVertexArray();
```

- Postavljanje aktivnog VAO objekta

```
gl.bindVertexArray(vao);
```

- Dobra praksa je deaktivirati VAO objekt nakon što smo u njega spremili sve potrebne informacije.

```
gl.bindVertexArray(null);
```

VAO objekti – primjer s isprepletinim bufferom

```
...
vertexBuffer = gl.createBuffer();
indeksBuffer = gl.createBuffer();
vao = gl.createVertexArray();

gl.bindVertexArray(vao);
gl.enableVertexAttribArray(program.a_position);
gl.enableVertexAttribArray(program.a_color);
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
gl.vertexAttribPointer(program.a_position,2,gl.FLOAT,false,20,0);
gl.vertexAttribPointer(program.a_color,3,gl.FLOAT,false,20,8);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vrhovi), gl.
    STATIC_DRAW);
gl.bindBuffer(gl.ARRAY_BUFFER, null);

gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indeksBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint8Array(indeksi), gl.
    STATIC_DRAW);
gl.bindVertexArray(null);
...
```

VAO objekti – važna napomena

- VAO objekt stanje `gl.ARRAY_BUFFER` buffera pamti preko `gl.enableVertexAttribArray` i `gl.vertexAttribPointer` metoda. Zato je bitno da se te metode pozovu nakon što smo aktivirali odgovarajući VAO objekt i prije nego što smo ga deaktivirali.
- VAO objekt stanje `gl.ELEMENT_ARRAY_BUFFER` buffera pamti preko `gl.bindBuffer` i `gl.bufferData` metoda. Zato je bitno da se te metode pozovu nakon što smo aktivirali odgovarajući VAO objekt i prije nego što smo ga deaktivirali. Također, unutar VAO objekta ne smijemo deaktivirati taj buffer jer ćemo ga onda onemogućiti u VAO objektu. Aktivacija takvog buffera je samo lokalna, tj. nije vidljiva izvan VAO objekta.

Iscrtavanje

- `gl.drawArrays` – za kreiranje objekta koristi podatke o vrhovima onim redom kako su spremljeni u jednom ili više buffera. ▸ gl.drawArrays
- `gl.drawElements` – za kreiranje objekta koristi indekse za pristup podacima o vrhovima koji su spremljeni u jednom ili više buffera. ▸ gl.drawElements
- Obje metode koriste samo trenutno omogućena polja, tj. buffer objekte koji su pridruženi aktivnim shader atributima.
- Više o osnovnim objektima koje `WebGL` zna crtati možete pogledati na ▸ WebGL2 fundamentals ili ▸ Learn WebGL.

Primjer 6.1. – WebGL kontekst

```
<script>
window.onload = initWebGL;
var gl = null;
function initWebGL() {
  var kan = document.getElementById("kan");
  gl = kan.getContext("webgl2");
  if (!gl) alert("WEBGL2 nije dostupan!");
  gl.clearColor(1, 0.6, 1, 1);
  gl.clear(gl.COLOR_BUFFER_BIT);
  console.log(gl);
  var boja = gl.getParameter(gl.COLOR_CLEAR_VALUE);
  console.log('clear color value:', boja);
}
</script>
```

Primjer 6.1. – WebGL kontekst

- Promijenite u kodu boju kojom će se obojati pozadina canvas elementa.
- Posjetite stranicu ▸ WebGL Report za dobivanje informacija o WebGL2 podršci u vašem web pregledniku. Pronađite informaciju o maksimalno dozvoljenom broju atributa u vertex shaderu.
- U priloženom primjeru otvorite konzolu i pogledajte informacije o WebGL2 kontekstu i pronađite informaciju o konstanti `MAX_VERTEX_ATTRIBS`. Pomoću `gl.getParameter` saznajte vrijednost te konstante i usporedite ju s vrijednošću dobivenoj na gornjoj web adresi.
- O WebGL2 kontekstu možete detaljnije pogledati, npr. na ▸ WebGL2 Context.

Crtanje linija i trokuta

Primjer 6.2.

- U primjeru je definirano 6 vrhova koji su pohranjeni u odgovarajući buffer.
- Pozivom metode `gl.drawArrays` s opcijom `gl.LINES` spojena su dva po dva vrha.
- Proučite kôd i riješite sve zadatke koji su napisani u `HTML` dokumentu koji se nalazi na moodlu.

Indeksiranje vrhova

Primjer 6.3.

- Kvadrat je nacrtan pomoću metode `gl.drawElements` i opcije `gl.TRIANGLE_FAN`. Kôd je dostupan na moodlu.
- Nacrtajte taj isti kvadrat pomoću metode `gl.drawElements` i opcije `gl.TRIANGLE_STRIP`.
- Nacrtajte taj isti kvadrat pomoću metode `gl.drawElements` i opcije `gl.TRIANGLES`.

Bojanje vrhova

Primjer 6.4.

- Kvadrat je nacrtan pomoću metode `gl.drawElements` i opcije `gl.TRIANGLES`. Kôd je dostupan na moodlu.
- Vrhovi imaju dva atributa: poziciju i boju. Svaki od atributa je spremljen u zasebni buffer.
- Pokazano je kako se boje vrhova iz vertex shadera prenose u fragment shader preko `varying` varijable gdje se onda obavlja interpolacija tih boja za dobivanje boja svih piksela od kojih se kvadrat sastoji.
- Mijenjajte boje vrhova tako da dobijete drukčije bojanje kvadrata. Pridružite nekim vrhovima iste boje.

Isprepleteni buffer

Primjer 6.5.

- Kvadrat je nacrtan pomoću metode `gl.drawElements` i opcije `gl.TRIANGLES`. Kôd je dostupan na moodlu.
- Vrhovi imaju dva atributa: poziciju i boju. Oba atributa su spremljena u isti buffer.
- Obratite pažnju na zadnja dva parametra metode `gl.vertexAttribPointer`.
- Rasporedite atribute u buffer tako da prvo napišete koordinate svih vrhova, a nakon toga unesete njihove boje. Prilagodite za taj slučaj parametre u metodi `gl.vertexAttribPointer`.

Boja ovisna o koordinatama

Primjer 6.6.

- Kvadrat je nacrtan pomoću metode `gl.drawElements` i opcije `gl.TRIANGLE_FAN`. Kôd je dostupan na moodlu.
- Primjer pokazuje kako iz vertex shadera možemo u fragment shader prenijeti koordinate svakog vrha i na temelju koordinata odrediti boju piksela prema nakoj matematičkoj formuli.
- Eksperimentirajte s raznim funkcijama u fragment shaderu za dobivanje različitih efektnih bojanja.

Reference

WebGL2 fundamentals

Learn WebGL