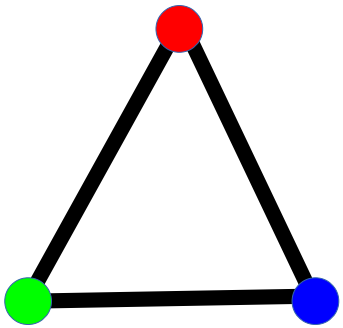
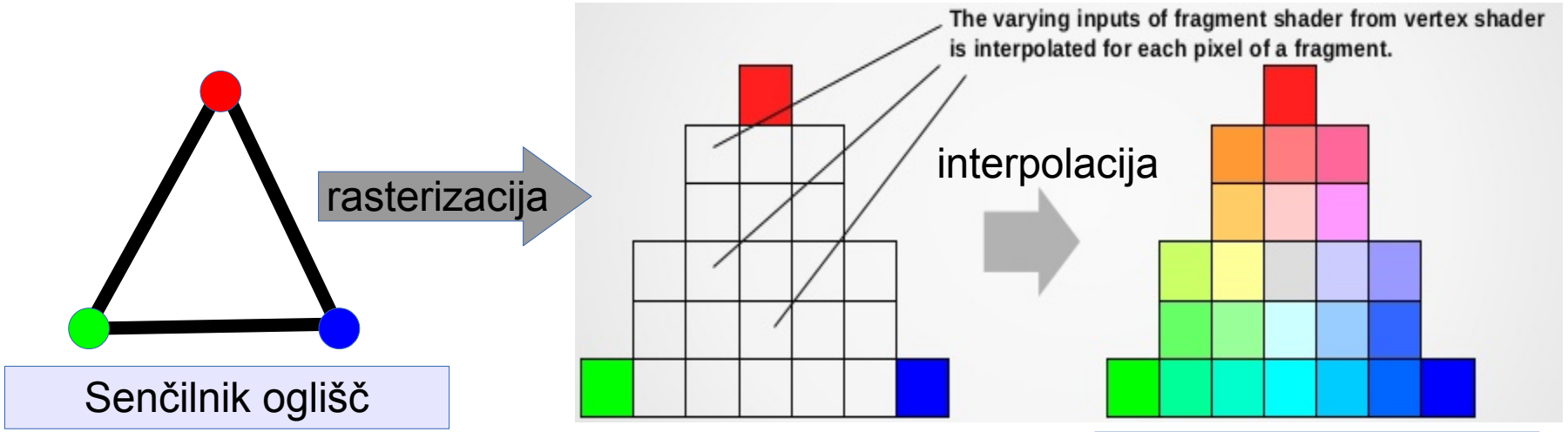
Dodatni podatki o ogliščih



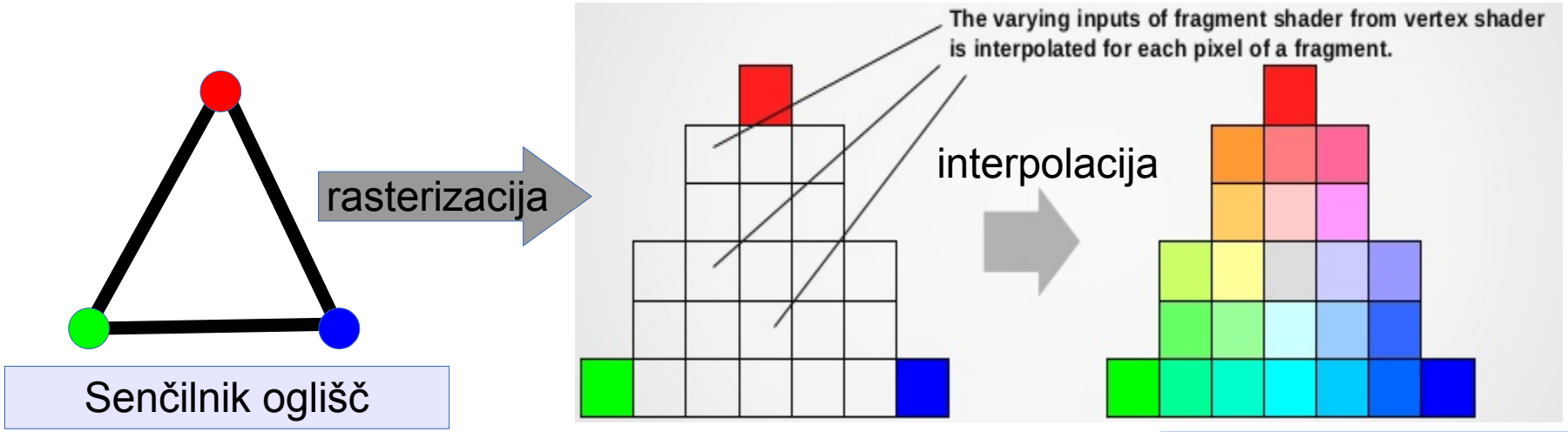
Senčilnik oglišč

Interpolacija



http://www.slideshare.net/champ_yen/opengl-es-2x-programming-introduction

Interpolacija (OpenGL 3.3 ->)



http://www.slideshare.net/champ_yen/opengl-es-2x-programming-introduction

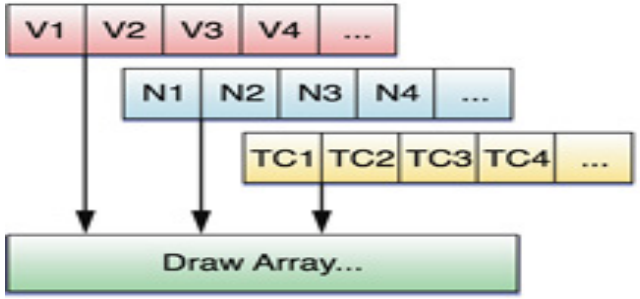
```
layout(location=0) in vec3 in_Pos;
layout(location=1) in vec4 in_Color;
out vec4 ColorFS;
...
void main(){
      ColorFS=in_Color;
...
```

```
in vec4 ColorFS;
out vec4 out_Color;

void main(void){
    out_Color=ColorFS;
...
```

Nalaganje dodatnih podatkov o ogliščih

- Za vsako oglišče lahko imamo dodatne podatke: barva, koordinate tekstur, normale. Zapis dodatnih lastnosti:
- Podatke lahko naložimo v ločena polja: (*glGenBuffers*, *glBindBuffer*)



http://www.cheetah3d.com/forum/showthread.php?t=9813

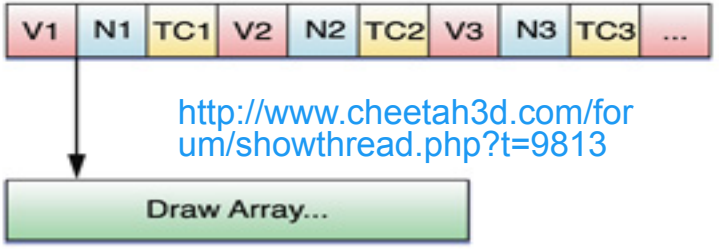
glGenVertexArrays, glBindVertexArray, ...
glCreateBuffer, glBindBuffer….., glBufferData
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(vec3), (int*)0);

glCreateBuffer, glBindBuffer….., glBufferData
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(vec3), (int*)0);

Nalaganje dodatnih podatkov o ogliščih

- Za vsako oglišče lahko imamo dodatne podatke: barva, koordinate tekstur, normale. Zapis dodatnih lastnosti:
- Podatke lahko zapišemo v eno polje (prepleten način, interleaved): najlažje

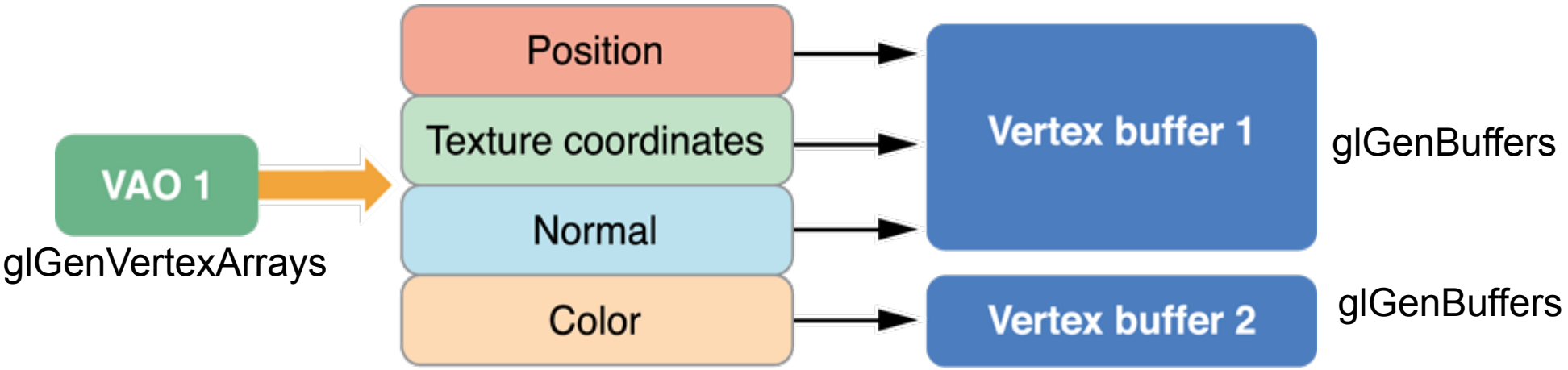
```
struct Tocka{
vec3 Position;
vec3 Color;
vec3 …;
};
```



glGenVertexArrays, glBindVertexArray, ...
glCreateBuffer, glBindBuffer….., glBufferData
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Tocka), (int*)offsetof(Tocka,Position));
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Tocka), (int*)offsetof(Tocka,Color));

Nalaganje dodatnih podatkov o ogliščih

- Možen je tudi hibridni pristop



https://developer.apple.com/library/ios/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/TechniquesforWorkingwithVertexData/TechniquesforWorkingwithVertexData.html

Uporaba dodatnih podatkov o ogliščih v senčilnikih

- Dodatni podatki: barve, koordinate tekstur, normale...
  - Te podatke ponavadi iz senčilnika oglišč pošljemo v senčilnik fragmentov preko rasterizacije (interpolacija)
- Potrebna povezava podatkov z vhodom v senčilnik oglišč:
  - Senčilnik oglišč: `layout(location=X) in vec3 in_pos;`
  - Prenos podatkov iz medpomnilnika na GPU do senčilnika oglišč:

```
struct Tocka{
 vec3 Position;
 vec3 Color;
 vec3 …;
};
```

```
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Tocka), (int*)offsetof(Tocka,Position));
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Tocka), (int*)offsetof(Tocka,Color));
```

Uvoz poljubnih objektov iz datoteke v grafični cevovod

- 3D modele oz. geometrijske podatke lahko dobimo iz datotek: https://free3d.com/ in jih prikažemo v naši aplikaciji
  - Najpogosteje uporabljan datotečni format: OBJ
- Uvoz 3D modelov:
  - https://github.com/assimp/assimp - knjižnica za uvažanje modelov iz različnih animacijskih paketov (Blender, Maya)
  - Enostavnejša možnost za uvažanje datotek OBJ: https://github.com/OpenGLInsights/OpenGLInsightsCode/blob/master/Chapter%2026%20Indexing%20Multiple%20Vertex%20Arrays/common/objloader.cpp

Primer datoteke *.obj

```
# Blender v2.76 (sub 0) OBJ File: ''
# www.blender.org
v -0.853250 -0.136293 -0.188932
v -0.129650 0.416492 0.336788
v -1.129635 0.416492 0.661708
v -1.747674 0.416492 -0.188932
v -1.129635 0.416492 -1.039572
v -0.129650 0.416492 -0.714652
v -0.576865 1.310922 0.661708
v -1.576849 1.310922 0.336788
v -1.576849 1.310922 -0.714652
v -0.576865 1.310922 -1.039572
v 0.041175 1.310922 -0.188932
v -0.853250 1.863707 -0.188932
```

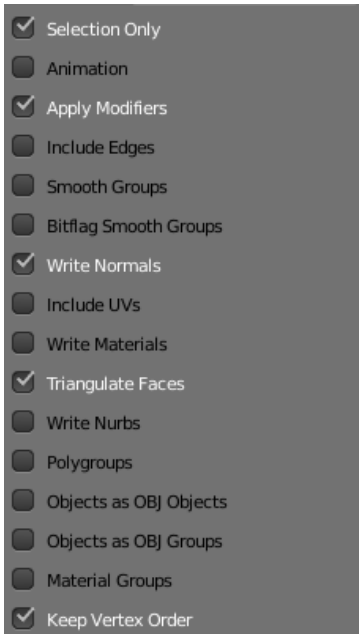
oglišča

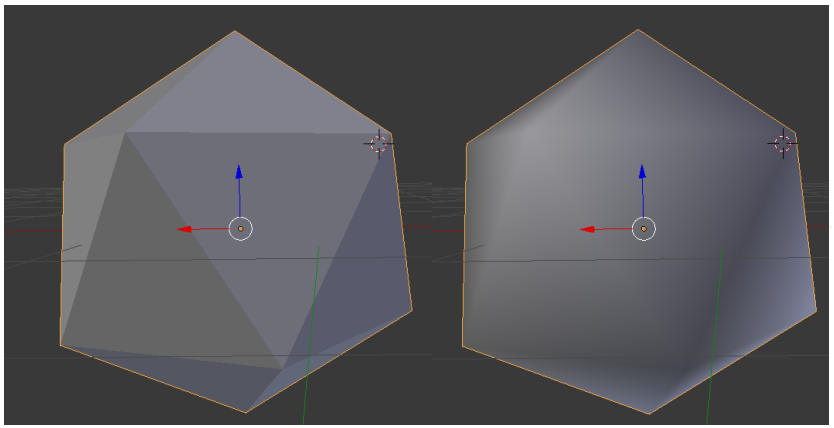
```
vt 0.500000 0.249994
vt 0.500000 0.749995
vt 0.404510 0.654504
vt 1.000000 0.250005
vt 1.000000 0.750006
vt 0.904509 0.654507
vt 0.250000 0.190977
vt 0.250001 0.000000
vt 0.750001 1.000000
vt 0.750000 0.190977
vt 0.249999 1.000000
vt 0.095491 0.654507
vt 0.000000 0.250005
vt 0.595490 0.654504
vt 0.749999 0.000000
vt 0.000000 0.750006
```

koordinate za teksture

```
vn 0.000000 -1.000000 0.000000
vn 0.723600 -0.447200 0.525700
vn -0.276400 -0.447200 0.850600
vn 0.723600 -0.447200 -0.525700
vn -0.894400 -0.447200 0.000000
vn -0.276400 -0.447200 -0.850600
vn 0.894400 0.447200 0.000000
vn 0.276400 0.447200 0.850600
vn -0.723600 0.447200 0.525700
vn -0.723600 0.447200 -0.525700
vn 0.276400 0.447200 -0.850600
vn 0.000000 1.000000 0.000000
```

normale





```
s 1
f 1/1/1 2/2/2 3/3/3
f 2/4/2 1/5/1 6/6/4
f 1/1/1 3/3/3 4/7/5
f 1/1/1 4/7/5 5/8/6
f 1/5/1 5/9/6 6/6/4
f 2/4/2 6/6/4 11/10/7
f 3/3/3 2/2/2 7/11/8
f 4/7/5 3/3/3 8/12/9
f 5/8/6 4/7/5 9/13/10
f 6/6/4 5/9/6 10/14/11
f 2/4/2 11/10/7 7/15/8
f 3/3/3 7/11/8 8/12/9
f 4/7/5 8/12/9 9/13/10
f 5/9/6 9/2/10 10/14/11
f 6/6/4 10/14/11 11/10/7
f 7/15/8 11/10/7 12/1/12
f 8/12/9 7/11/8 12/16/12
f 9/13/10 8/12/9 12/16/12
f 10/14/11 9/2/10 12/1/12
f 11/10/7 10/14/11 12/1/12
```

trikotniki:

f indeks oglisca1/tekstura1/normala1
indeks oglisca2/tekstura2/normala2
indeks oglisca3/tekstura1/normala3

Postopek branja datoteke OBJ

1) Zaporedno branje položajev oglišč, normal, koordinat tekstur v std::vector<glm::vec3/glm::vec2>
    1) Vrstice se začnejo na: v (vec3), vt (vec2), vn (vec3)
2) Zaporedno branje indeksov iz trikotnikov
    1) Vrstice se začnejo z znakom »f« (faces)
    2) Preberemo indekse
    3) Preko indeksov uporabimo std::vector oglišč, normal in koordinat tekstur iz 1. koraka
        1) Indeksi se začnejo z 1!
    4) Zapisovanje oglišč v novi std::vector trikotnikov
        1) Za vsako oglišče zapišemo tri vektorje
    5) **Nalaganje std::vector na GPU**

Prikaz več geometrijskih objektov v OpenGL

- Želimo prikaz poljubnega števila geometrijskih objektov / 3D modelov!
- Za vsak objekt hranimo:
  - položaj, rotacija, skaliranje,
  - podatke o geometrijski predstavitvi,
  - ostale podatke.
- Implementacija
  - Celotno sceno predstavimo s seznamom objektov, npr: std::vector<Objekt>, v vsak objekt zapišemo ustrezne informacije
    - V OpenGL za vsak geometrijski objekt potrebujemo število oglišč (za glDrawArrays), VAO in VBO
  - Za vsak geometrijski objekt ob inicializaciji kličemo **glGenVertexArrays**, **glBindVertexArray**, **glCreateBuffer, glBindBuffer, glBufferData**, **glVertexAttribPointer**
  - Ob izrisu kličemo zgolj **glBindVertexArray, glUniform..., glDrawArrays**
    - https://registry.khronos.org/OpenGL-Refpages/gl4/html/glUniform.xhtml

Kombiniranje OpenGL z grafičnim vmesnikom aplikacije

- Qt in podobne knjižnice (naprednejše) imajo sam izris implementiran tudi v OpenGL
- Vsako okno ima ponavadi svoj kontekst od OpenGL (lastno stanje grafičnega cevoda)
- V primeru prikaza novih oken (dialog za nalaganje datotek - http://doc.qt.io/qt-5/qfiledialog.html) pazite, da pred novimi klici OpenGL nad widgetRisanje kličete openGLwidget->makeCurrent()
  - http://doc.qt.io/qt-5/qopenglwidget.html#makeCurrent
  - Ta funkcija določa, da se vsi nadaljnji klici od OpenGL izvajajo nad QOpenGLWidget (QOpenGLContext) in ne nad dialogom za odpiranje datotek.
  - V nasprotnem primeru spreminjate izris novega okna!