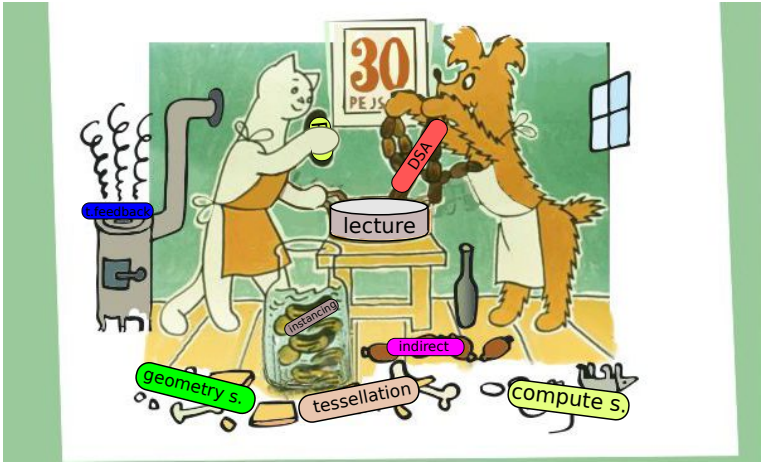
Advanced OpenGL

Tomáš Milet

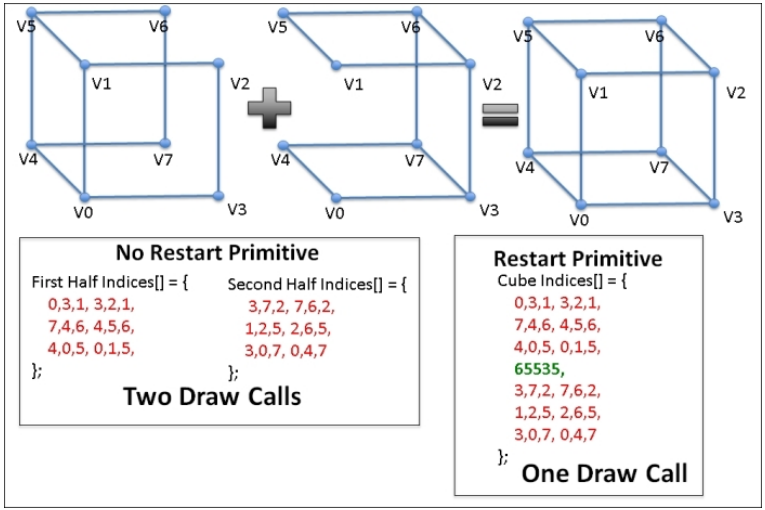
Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole
imilet@fit.vutbr.cz

22. října 2021

Content of this lecture



Primitive restart index



https://www.packtpub.com/mapt/book/application_development/9781849695527/3/ch03lvl1sec36/

Primitive restart index

- Primitive restart index can be use to terminated triangle strip.
- If we want to draw a lot of triangle strips that are not connected, we can use PRI.
- Without PRI, we would have to use glDrawElements for each triangle strip.
- or we would have to degenerated some primitives.
- PRI is used as element of element array buffer.
- Primitive assembly is terminated after reaching PRI.

```
glEnable(GL_PRIMITIVE_RESTART);
glPrimitiveRestartIndex(0xffffffff);
```

Advanced draw commands

- There is a lot of variants of following commands:

```
glDrawArrays glDrawElements
```

- Instancing
- Indirect Draw
- Multi Draw

```
glDrawArrays,glMultiDrawArrays

glDrawArraysInstanced,glDrawArraysInstancedBaseInstance,

glDrawArraysIndirect,glMultiDrawArraysIndirect

glDrawElements,glDrawRangeElements,glMultiDrawElements

glDrawElementsBaseVertex,glDrawRangeElementsBaseVertex,glMultiDrawElementsBaseVertex
glDrawElementsInstanced,glDrawElementsInstancedBaseInstance,glDrawElementsInstancedBaseVertex,
glDrawElementsInstancedBaseVertexBaseInstance

glDrawElementsIndirect,glMultiDrawElementsIndirect
```

Advanced draw commands - instancing

http://epicbattles.wikia.com/wiki/File:UEBS_Chunk.jpg

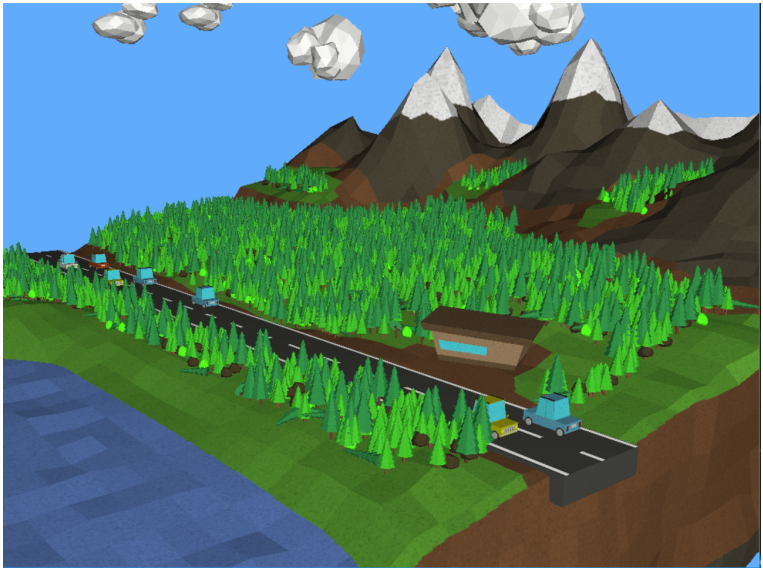
Advanced draw commands - instancing

- Instancing draws same mesh in multiple instances, each with different transformation matrix.
- Instance id can be access in shader through built-in variable: gl_InstanceID
- Instance id can be used as index into array of transformation matrices or materials.

```
#version 460

layout(location=0)in vec4 Position;
uniform mat4 MVP[100];

void main(){
  gl_Position=MVP[gl_InstanceID]*Position;
}
```

```
glBindVertexArray(VAO);
glDrawArraysInstanced(GL_TRIANGLES,0,NumVertices,NumInstances);
glBindVertexArray(0);
```

Advanced draw commands - indirect calls



Advanced draw commands - indirect draw calls

- Draw call can be stored in buffers.
- There is no need for CPU synchronization.
- It can be combined with compute shader, atomic counters, transform feedback, ...

```
//init
glGenBuffers(1,&IndirectBuffer);
glBindBuffer(GL_DRAW_INDIRECT_BUFFER,IndirectBuffer);
unsigned Data[4]={100,1,0,0};//command
glBufferData(GL_DRAW_INDIRECT_BUFFER,sizeof(unsigned)*4,
  Data,GL_DYNAMIC_DRAW);

//...
//fill buffer from GPU
//...

//draw
glBindBuffer(GL_DRAW_INDIRECT_BUFFER,IndirectBuffer);
glDrawArraysIndirect(GL_TRIANGLES,NULL);
```

Advanced draw commands - multi draw calls

- Multi draw call fuses a lot of single draw calls into one.
- All draw calls parameters can be stored in buffers and generated on GPU.
- Frustum Culling on GPU - number of instances of visible object objekt ¿ 0

```
//init
glGenBuffers(1,&IndirectBuffer);
glBindBuffer(GL_DRAW_INDIRECT_BUFFER,IndirectBuffer);
glBufferData(GL_DRAW_INDIRECT_BUFFER,sizeof(unsigned)*5*NumSpheres,
  NULL,GL_DYNAMIC_COPY);
//...
glDispatchCompute(NumSpheres/WorkGroupSize.x+1,1,1);
//...
//draw
glBindVertexArray(VAO);
glBindBuffer(GL_DRAW_INDIRECT_BUFFER,IndirectBuffer);
glMultiDrawElementsIndirect(GL_TRIANGLES,GL_UNSIGNED_INT,NULL,
  NumSpheres,sizeof(unsigned)*5);
glBindVertexArray(0);
```

Bindless textures

- We often need to draw a lot of objects with different textures.
- This leads to texture switching in between draw commands and to greater number of draw commands.
- Texture binding is not cheap and there is limited number of texture units.
- Texture atlas has its own problems (mipmapping, color bleeding, ...)
- There is an extension: ARB_bindless_texture that solves a lot of problems
- This extension allows shader to switch textures directly.
- Textures are not explicitely bind to any texture unit.

Bindless textures - handles

```
#define MAX_TEXTURES 512
GLuint textures[MAX_TEXTURES];//list of textures
glCreateTextures(GL_TEXTURE_2D,MAX_TEXTURES,textures);//create textures
for(unsigned i=0;i<MAX_TEXTURES;++i){//loop over textures
  glTextureStorage2D(textures[i],1,GL_RGBA32F,1024,1024);//texture allocation
  glTextureSubImage2D(textures[i],0,0,0,1024,1024,
    GL_RGBA,GL_UNSIGNED_BYTE,data[i]);//data upload
  ...

}
```

```
GLuint64 handles[MAX_TEXTURES];//list of handles to textures
for(unsigned i=0;i<MAX_TEXTURES;++i){
  handles[i]=glGetTextureHandleARB(textures[i]);
  glMakeTextureHandleResidentARB(handles[i]);
}
```

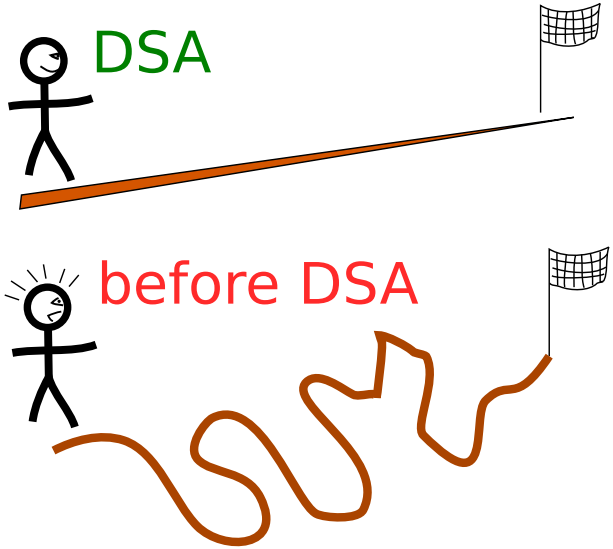
```
glProgramUniformHandleui64vARB(
    programId,//program id
    uniformLocation,//location of uniform variable
    MAX_TEXTURES,//number of handles
    handles);//handles
```

Bindless textures - vertex shader

```
#version 440 core

#extension GL_ARB_bindless_texture : require

#define MAX_TEXTURES 512

layout(bindless_sampler)uniform sampler2D textures[MAX_TEXTURES];
flat out sampler2D sampler;

struct Material{
  uint  textureId;
  vec3  color;
};

layout(std430,binding=0)buffer MaterialArray{Material materials[];};

layout(location=3) in uint materialID;

void main(){
  sampler = textures[materials[materialID]];
  gl_Position=...;
}
```

Bindless textures - fragment shader

```
#version 440 core

#extension GL_ARB_bindless_texture : require

layout(location=0)out vec4 fColor;

flat in sampler2D sampler;

in vec2 vTexCoord;

void main(){
  fColor=texture(sampler,vTexCoord);
}
```


DSA
before DSA

Direct State Access

- Since OpenGL 4.5, Khronos added large extension into core profile - Direct State Access.
- DSA adds a lot of new functions for direct manipulation of objects.
- Bind to modify paradigm will be deprecated in the future.

Old approach:

1. Backup object id that is currently bound to target.
2. Bind new object to target.
3. Execute operation on target.
4. Rebind backup to target.

New approach:

1. Execute operation on object.

Direct State Access

- DSA is easier and follow common sense.
- DSA does not break bindings.
- DSA is faster, less state switching.
- Target is replaced with object id in most OpenGL functions.
- DSA exists as extension for a long time - it is supported on all GPUs

Direct State Access - examples

Bind to modify:

```
GLint old;
glGenRenderbuffers(1,&RBO);
glGetIntegerv(GL_RENDERBUFFER_BINDING,&old);
glBindRenderbuffer(GL_RENDERBUFFER,RBO);
glRenderbufferStorage(GL_RENDERBUFFER,internalFormat,width,height);
glBindRenderbuffer(GL_RENDERBUFFER,old);
```

Direct state access:

```
glCreateRenderbuffers(1,&RBO);
glNamedRenderbufferStorage(RBO,internalFormat,width,height);
```

Direct State Access - examples

Bind to modify:

```
glBindFramebuffer(GL_READ_FRAMEBUFFER,fbo);
glBindFramebuffer(GL_DRAW_FRAMEBUFFER,0);
glBlitFramebuffer(0,0,width,height,0,0,width,height,
  GL_COLOR_BUFFER_BIT,GL_NEAREST);
```

Direct state access:

```
glBlitNamedFramebuffer(fbo,0,0,0,width,height,0,0,width,height,
  GL_COLOR_BUFFER_BIT,GL_NEAREST);
```

Direct State Access - confusion

- Some OpenGL action have 2 or even 3 alternative commands.
- A lot of functions is redundant.
- Confusion in texture, vao commands

```
glFramebufferTexture
glFramebufferTexture1D;
glFramebufferTexture2D;
glFramebufferTexture3D;
glFramebufferTextureLayer;
glNamedFramebufferTexture;
glNamedFramebufferTextureLayer;
```

```
glVertexArrayBindingDivisor
glVertexAttribDivisor
glVertexBindingDivisor
```

```
glTexImage2D
glTexStorage2D
glTextureStorage2D
```

Program branching, functionality switching

- Graphics application contains a lot of different effects (bump mapping, shadows, paralax mapping, ...)
- Different effects require different shaders.
- Functionality switching can be performed on different levels.
- Switching program path using uniform variable.
- Subroutines switching
- Shader program pipelines switching
- Shader program switching

Functionality switching - uniformn variables

The fastest functionality switch can be performed directly inside shader. However, there is per-invocation overhead. For example, vertex shader has to evaluate condition for every vertex (thread divergence).

```
#version 430
uniform uint method;
void main(){
  switch(method){
    case 1:
      normalMapping(...);
      break;
    case 2:
      paralaxMapping(...);
      break;
    case 3:
      ...
  }
}
```

It is the second fastest functionality switch. It is function pointer switching.

```
#version 440

layout(location=0)out vec4 fColor;

subroutine vec4 getColorSubroutine();
subroutine vec4 rotateColorSubroutine(vec4);

subroutine (getColorSubroutine) vec4 redColor(){return vec4(1,0,0,1);}
subroutine (getColorSubroutine) vec4 greenColor(){return vec4(0,1,0,1);}

subroutine (rotateColorSubroutine) vec4 rotate1Left (vec4 c){return c.yzwx;}
subroutine (rotateColorSubroutine) vec4 rotate2Left (vec4 c){return c.zwxy;}
subroutine (rotateColorSubroutine) vec4 rotate3Left (vec4 c){return c.wxyz;}
subroutine (rotateColorSubroutine) vec4 reverse     (vec4 c){return c.wzyx;}

subroutine uniform getColorSubroutine    getColor;//fce pointer
subroutine uniform rotateColorSubroutine rotateColor[3];//array of fce pointers

uniform uint rotateIndex=0;

void main(){
  fColor=rotateColor[rotateIndex](getColor());
}
```

Functionality switching - subroutines

```
//all subroutines
GLuint sl0=glGetSubroutineIndex(program,GL_FRAGMENT_SHADER,"redColor");
GLuint sl1=glGetSubroutineIndex(program,GL_FRAGMENT_SHADER,"greenColor");
GLuint sl2=glGetSubroutineIndex(program,GL_FRAGMENT_SHADER,"rotate1Left");
GLuint sl3=glGetSubroutineIndex(program,GL_FRAGMENT_SHADER,"rotate2Left");
GLuint sl4=glGetSubroutineIndex(program,GL_FRAGMENT_SHADER,"rotate3Left");
GLuint sl5=glGetSubroutineIndex(program,GL_FRAGMENT_SHADER,"reverse");

//uniform offsets for subroutines
GLuint sul0=glGetSubroutineUniformLocation(program,GL_FRAGMENT_SHADER,"getColor");
GLuint sul1=glGetSubroutineUniformLocation(program,GL_FRAGMENT_SHADER,"rotateColor");

glUseProgram(program);
//list of all slots for subroutines
GLuint sl[4];
sl[sul0+0]=sl1;
sl[sul1+0]=sl5;
sl[sul1+1]=sl2;
sl[sul1+2]=sl3;
glUniformSubroutinesuiv(GL_FRAGMENT_SHADER,4,sl);
```

Functionality switching - pipelines

- Program pipelines can be used for shader stage switching (vertex,fragment,geometry,...).
- Inputs and outputs of stages has to match, they will not be checked.
- Pipeline switching is cheaper that program switching.
- It is usefull in situation where there is a lot of similar shader programs.

```
GLuint vs,gs,fs;
//This function creates shader program that contains only one shader stage.
vs=glCreateShaderProgramv(GL_VERTEX_SHADER,1,&vstext);
fs=glCreateShaderProgramv(GL_FRAGMENT_SHADER,1,&fstext);
//...

GLuint pipeline;
glGenPipelines(1,&pipeline);
glBindPipelines(pipeline);

//...
glUseProgramStages(pipeline,GL_VERTEX_SHADER_BIT,vs);
glUseProgramStages(pipeline,GL_GEOMETRY_SHADER_BIT,program);//select geometry shader from shader program
glUseProgramStages(pipeline,GL_FRAGMENT_SHADER_BIT,fs);
```

Geometry shader

- Geometry shader is located after vertex shader (after tessellation).
- Geometry shader is executed per primitive - it has access to all vertices of a primitive.
- It can generate new geometry or modify current geometry.
- It can transform point into quad (usefull for particle simulation).
- It can be used for several effects (shadow volumes, particle systems, ...).
- Geometry Instancing.
- Transform feedback.

Geometry shader - inputs/outputs

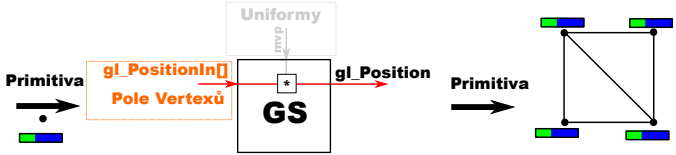
- Type of inputs and outputs has to be specified inside geometry shader.

```
layout(points,invocations=N)in;//input primitive will be point.
//points, lines, lines_adjacency, triangles, triangles_adjacency
//geometry shader will be executed N times on each point.
```

- Type of output primitive and number of output vertices has to be specified.

```
layout(triangle_strip,max_vertices=4)out;//output primitive and max number of vertices
//points, line_strip, triangle_strip
```

Geometry shader - point to quad

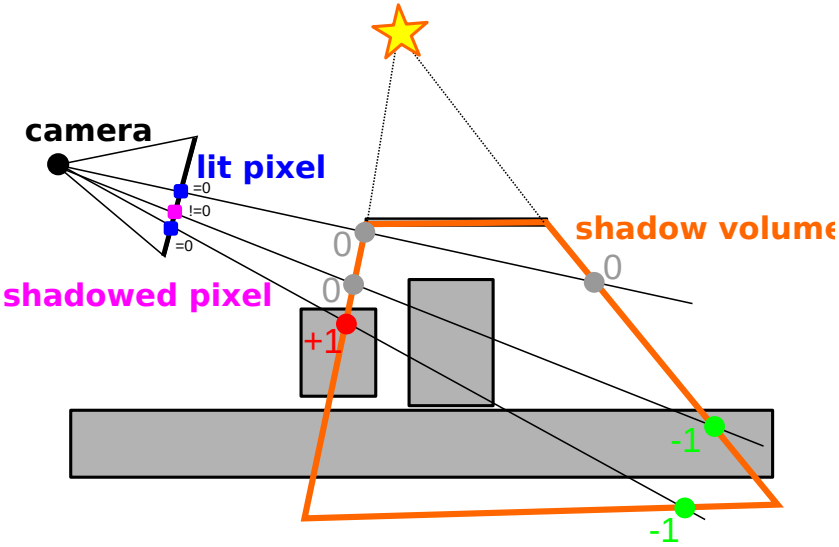



```
#version 430

layout(points)in;
layout(triangle_strip,max_vertices=4)out;

void main(){
  gl_Position=mvp*(gl_in[0].gl_Position+vec4(-1,-1,0,0));
  EmitVertex();
  gl_Position=mvp*(gl_in[0].gl_Position+vec4(-1,+1,0,0));
  EmitVertex();
  gl_Position=mvp*(gl_in[0].gl_Position+vec4(+1,-1,0,0));
  EmitVertex();
  gl_Position=mvp*(gl_in[0].gl_Position+vec4(+1,+1,0,0));
  EmitVertex();
  EndPrimitive();
}
```

Geometry shader - fullscreen quad

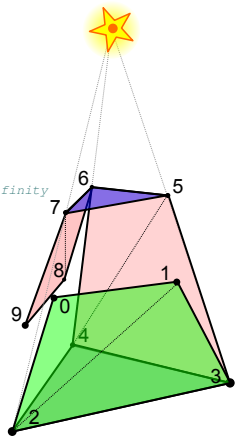
```
#version 430
layout(points)in;
layout(triangle_strip,max_vertices=4)out;
void main(){
  gl_Position=vec4(-1,-1,0,1);EmitVertex();
  gl_Position=vec4(-1,+1,0,1);EmitVertex();
  gl_Position=vec4(+1,-1,0,1);EmitVertex();
  gl_Position=vec4(+1,+1,0,1);EmitVertex();
  EndPrimitive();
}
```

```
glGenVertexArrays(1,&emptyVAO);
//...
glBindVertexArray(emptyVAO);//activate VAO
glDrawArrays(GL_POINTS,0,1);
glBindVertexArray(0);//deactivate VAO
```


camera
lit pixel
=0
!=0
=0
shadowed pixel
shadow volume
0
0
0
+1
-1
-1

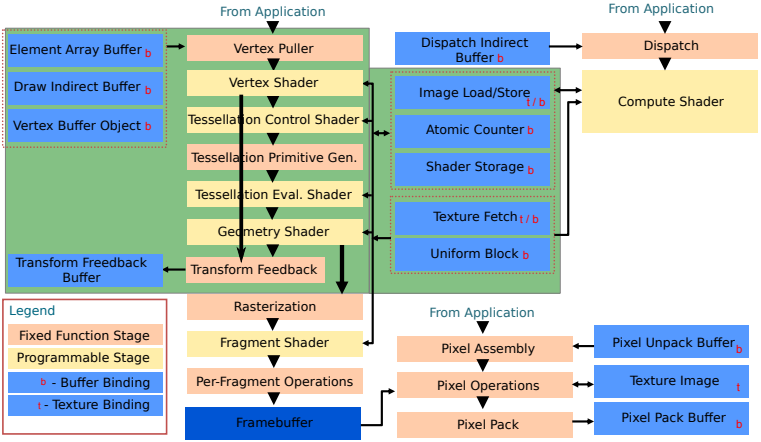
Geometry shader - Shadow volumes

```
#version 330
layout(triangles)in;
layout(triangle_strip,max_vertices=10)out;
uniform mat4 MVP,M;//matrices
uniform vec4 LightPosition;//light position
void main(){
  vec4 LP=M*LightPosition;
  vec4 p[6];
  p[0]=gl_in[0].gl_Position;//triangle vertices
  p[1]=gl_in[1].gl_Position;
  p[2]=gl_in[2].gl_Position;
  p[3]=vec4(gl_in[0].gl_Position.xyz*LP.w-LP.xyz,0);//triangle points projected to infinity
  p[4]=vec4(gl_in[1].gl_Position.xyz*LP.w-LP.xyz,0);
  p[5]=vec4(gl_in[2].gl_Position.xyz*LP.w-LP.xyz,0);
  vec3 N=normalize(cross((p[1]-p[0]).xyz,(p[2]-p[0]).xyz));
  float Distance=dot(N,LP.xyz)-dot(N,p[0].xyz);
  if(Distance<=0){//flip volume inside out
    vec4 c=p[0];p[0]=p[1];p[1]=c;
    c=p[3];p[3]=p[4];p[4]=c;
  }
  gl_Position=MVP*p[0];EmitVertex();
  gl_Position=MVP*p[1];EmitVertex();
  gl_Position=MVP*p[3];EmitVertex();
  gl_Position=MVP*p[4];EmitVertex();
  gl_Position=MVP*p[5];EmitVertex();
  gl_Position=MVP*p[1];EmitVertex();
  gl_Position=MVP*p[2];EmitVertex();
  gl_Position=MVP*p[0];EmitVertex();
  gl_Position=MVP*p[5];EmitVertex();
  gl_Position=MVP*p[3];EmitVertex();
  EndPrimitive();
}
```

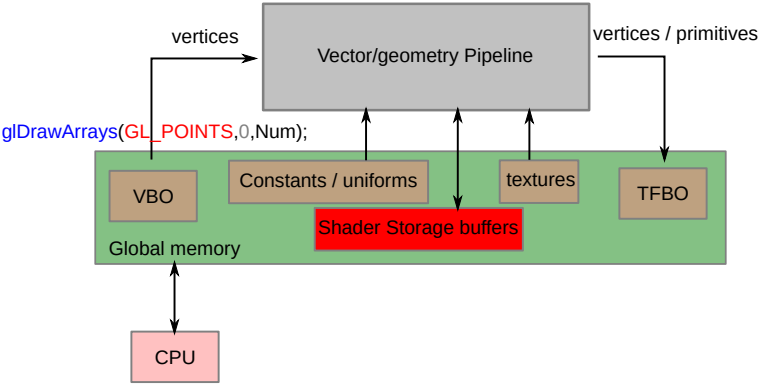


Transform feedback

- Transform feedback stops rendering pipeline before rasterisation and sends primitives into buffers.
- It can be used in vertex shader and geometry shader.
- It can be combined with rendering (streams).



Transform feedback



Transform feedback - example

```
const char*Vayrings[]={"Out1", "Out2"};
glTransformFeedbackVaryings(Program,2,Varyings,GL_SEPARATE_ATTRIBS);
glLinkProgram(Program);

//...

glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER,0,Buffer1);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER,1,Buffer2);

glEnable(GL_RASTERIZER_DISCARD);//nebudeme rasterizovat
//...
glBeginTransformFeedback(GL_TRIANGLES);
glDrawArrays(...);
glEndTransformFeedback();
```

We have to link shader program with marked varyings. c++:

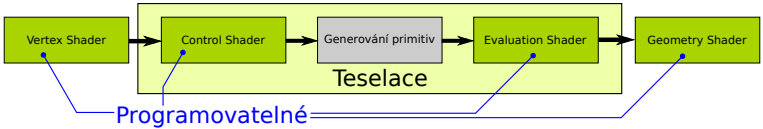
```
//list of variables that will be written into buffer(s).
const char*ResetVaryings[]={"vPosition","vVelocity","vMass"};
//set varyings and interleaving
glTransformFeedbackVaryings(ResetProgram,3,ResetVaryings,GL_INTERLEAVED_ATTRIBS);
//relink shader program
glLinkProgram(ResetProgram);
```

glsl:

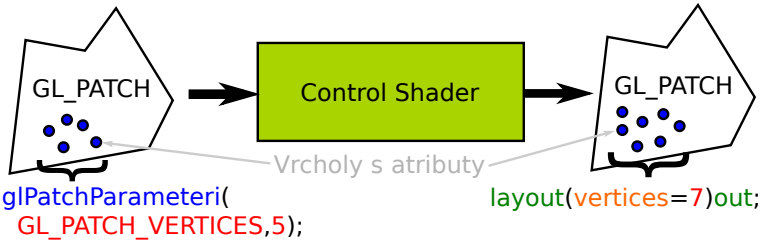
```
#version 330

layout(location=0)out vec2  vPosition;//particle position
layout(location=1)out vec2  vVelocity;//particle velocity
layout(location=2)out float vMass;//particle mass
//...
void main(){
  vPosition = vec2(0);//init position
  vVelocity = vec2(cos(VelAngle),sin(VelAngle))*VelSize;//init velocity
  vMass     = Noise(MassSeed+uint(gl_VertexID),MinMass,MaxMass);//init mass
}
```

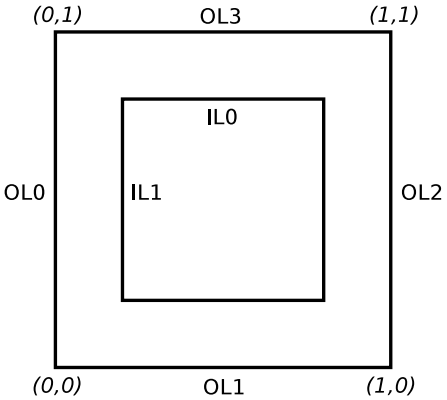
- Tessellation splits one primitive into more joint sub primitives.
- It can be used to refine details of a geometry.
- Tessellation is located between vertex shader and geometry shader.
- It is composed of three parts:
  - Control Shader
  - Primitive generation/tessellation
  - Evaluation Shader
- There is one new primitive type - GL_PATCHES

```
glPatchParameteri(GL_PATCH_VERTICES,10);//set number of vertices of patch to 10
glDrawArrays(GL_PATCHES,0,200);//draw 20 patches (each is composed of 10 vertices)
```

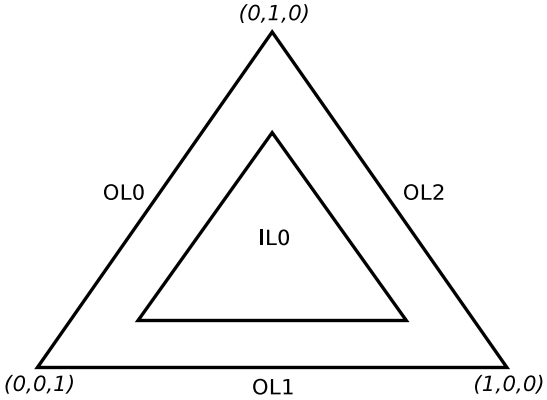


- CS controls level of tessellation.
- It computes control vertices.
- CS is invocated as many times as there is output vertices in output patch primitive.
- A invocation number is stored in built-in variable gl_InvocationID.
- We can synchronise threads in CS using barrier().





Quads



Triangles

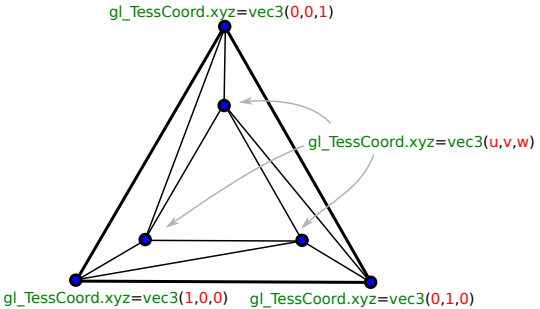
- layout ({isolines,triangles,quads}) in;
- gl_TessLevelOuter(4),gl_TessLevelInner(2)

Control Shader - example

```
#version 430

// number of output vertices in output patch primitive ==
// number of threads per patch primitive
layout(vertices=3)out;

uniform vec2 TessLevelInner;//there are two inner and
uniform vec4 TessLevelOuter;//four outer levels of tessellation

void main(){
  //number of elements in gl_in depends on GL_PATCH_VERTICES
  //number of elements in gl_out delends on layout(vertices=n)out;
  //In this case, number of elements in gl_in and gl_out is the same.
  gl_out[gl_InvocationID].gl_Position=gl_in[gl_InvocationID].gl_Position;//copy
  if(gl_InvocationID==0){//the first thread sets tessellation levels
    gl_TessLevelOuter[0]=TessLevelOuter[0];
    gl_TessLevelOuter[1]=TessLevelOuter[1];
    gl_TessLevelOuter[2]=TessLevelOuter[2];
    gl_TessLevelOuter[3]=TessLevelOuter[3];
    gl_TessLevelInner[0]=TessLevelInner[0];
    gl_TessLevelInner[1]=TessLevelInner[1];
  }
}
```

Evaluation Shader

- It sets type of tessellated primitive: isolines,triangles,quads
- It computes coordinates of tessellated vertices.
- gl_TessCoord variable holds barycentric, uv, or normalized coords of tessellated vertices inside of primitive.
- Evaluation shader is executed for every tessellated vertex.
- Tessellated primiteves continue their path to geometry shader.



Evaluation Shader - example

- Tessellated quad
- Computation of coordinates of tessellated vertices

```
#version 430

layout(quads)in;

void main(){
  vec4 A=mix(gl_in[0].gl_Position,gl_in[1].gl_Position,gl_TessCoord.x);
  vec4 B=mix(gl_in[3].gl_Position,gl_in[2].gl_Position,gl_TessCoord.x);
  gl_Position=mix(A,B,gl_TessCoord.y);
}
```

Béziér surface - example

```
// Vertex shader
#version 430
void main() {
  gl_Position = mvp*position;
}

// Control shader
#version 430
layout(vertices=16) out;

void main() {
  gl_out[gl_InvocationID].gl_Position =
  gl_in[gl_InvocationID].gl_Position;
  if(gl_InvocationID == 0) {
    gl_TessLevelInner[0] = gl_TessLevelInner[1] =
    gl_TessLevelOuter[0] = gl_TessLevelOuter[1] =
    gl_TessLevelOuter[2] = gl_TessLevelOuter[3] = 64;
  }
}
```

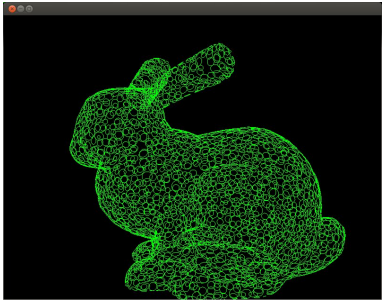
Béziér surface - example

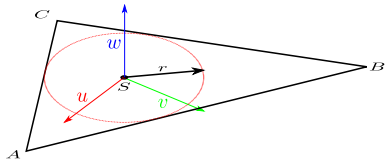
```
// Evaluation shader
#version 430
layout(quads, ccw) in;

vec4 bernstein(float t) {
  return vec4((1-t)*(1-t)*(1-t), 3*t*(1-t)*(1-t), 3*t*t*(1-t), t*t*t);
}

void main() {
  vec4 bu = bernstein(gl_TessCoord.x);
  vec4 bv = bernstein(gl_TessCoord.y);
  vec4 position = vec4(0, 0, 0, 0);
  for(int y = 0; y < 4; ++y){
    for(int x = 0; x < 4; ++x){
      position += bu[x]*bv[y]*gl_in[4*y + x].gl_Position;
    }
  }
  gl_Position = position;
}
```

Communication between shader stages

```
#version 430

//output vertex shader attributes
out vec4 vAttrib;
gl_Position

//intput control shader attributes
in  vec4 vAttrib[];//attribute from vertex shader, size == GL_PATCH_VERTICES
gl_in[].gl_Position;//a possition attribute from vertex shader

//output control shader attributes
out vec4 cAttrib[];//size == layout(vertices=size)out
patch out mat4 cM;//once per output patch primitive

//input evaluation shader attributes
in vec4 cAttrib[];//size == layout(vertices=size)out
patch in  mat4 cM;//once per input patch

//output evaluation shader attributes
out vec3 eNormal;

//input geometry shader attributes
in vec3 eNormal[];//size == 2 for line, size == 3 for triangle, ...
```

Example - replace triangles with circles





$$K = \begin{pmatrix} 1 & 0 & 0 & S_x \\ 0 & 1 & 0 & S_y \\ 0 & 0 & 1 & S_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r & 0 & 0 & 0 \\ 0 & r & 0 & 0 \\ 0 & 0 & r & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Example - replace triangles with circles

Control Shader

```
#version 400
layout(vertices=1)out;
patch out mat4 K;
void main(){
  gl_TessLevelOuter[0]=1;
  gl_TessLevelOuter[1]=64;
  gl_TessLevelOuter[2]=1;
  gl_TessLevelOuter[3]=1;
  gl_TessLevelInner[0]=1;
  gl_TessLevelInner[1]=1;
  vec4 TT[3];
  TT[0]=gl_in[0].gl_Position;
  TT[1]=gl_in[1].gl_Position;
  TT[2]=gl_in[2].gl_Position;
  float t01=length((TT[0]-TT[1]).xyz);
  float t02=length((TT[0]-TT[2]).xyz);
  float t12=length((TT[1]-TT[2]).xyz);
  float s=t01+t02+t12;
  float r=sqrt((s/2-t01)*(s/2-t02)*(s/2-t12)*s/2)*2/s;
  t01/=s;
  t02/=s;
  t12/=s;
  vec3 C=TT[0].xyz*t12+TT[1].xyz*t02+TT[2].xyz*t01;
  vec3 x=normalize(TT[0].xyz-C);
  vec3 y=normalize(TT[1].xyz-C);
  vec3 z=normalize(cross(x,y));
  y=normalize(cross(z,x));
  K=mat4(vec4(x,0)*r,vec4(y,0)*r,vec4(z,0)*r,vec4(C,1));
}
```

Evaluation Shader

```
#version 400

#define MY_PI 3.14159265359

layout(isolines)in;

uniform mat4 V;
uniform mat4 P;

patch in mat4 K;

void main(){
  float Angle=gl_TessCoord.x*MY_PI*2;
  vec4 PP=vec4(cos(Angle),sin(Angle),0,1);
  gl_Position=P*V*K*PP;
}
```

How to choose tessellation levels?

Outer level:

- Outer levels of edges of neighbor faces should be the same to prevent T-joints.
- Transform control points to the screen (screen-space distances).
- Divide edge length by maximal edge length.

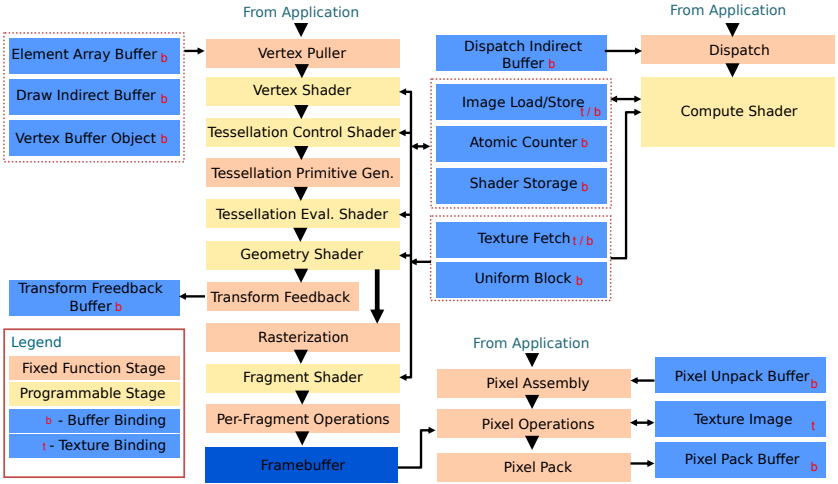
Inner level:

- Average/maximum of outer levels
- ...

Image Load/Store

- Reads/Writes from/to images inside of shader stage.
- There is new data type - image.
- An image is one layer of texture (one layer of mipmap, ...).
- There are atomic operation for images.
- Atomic operation are supported only for certain internal formats (integer, one channel).
- Image units (lequal 8 units) - simillar to texture units (gequal 80 units).
- Store operations are side effect. They disable early fragment tests in fragment shader - it can be reenabled.

Image Load/Store



This fragment shader emulates stencil buffer:

```
#version 430

layout(location=0)out vec4 fColor;
layout(early_fragment_tests)in;//enable early fragment tests
layout(r32i,location=0)uniform iimage2D myStencil;//integer image
ivec2 Coord=ivec2(gl_FragCoord.xy);//coordinates

void main(){
  if(gl_FrontFacing)//front face
    imageAtomicAdd(myStencil,Coord,+1);
  else
    imageAtomicAdd(myStencil,Coord,-1);
}
```

Image Load/Store - example

```
//integer texture
glGenTextures(1,&Image);
glBindTexture(GL_TEXTURE_2D,Image);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
glTexImage2D(GL_TEXTURE_2D,0,GL_R32I,Widht,Height,0,GL_RED_INTEGER,
  GL_UNSIGNED_BYTE,NULL);

//bind one layer of the texture onto zeroth image unit
glBindImageTexture(0,Image,0,GL_FALSE,0,GL_READ_WRITE,GL_R32I);
```

Atomic counters/Shader Storage

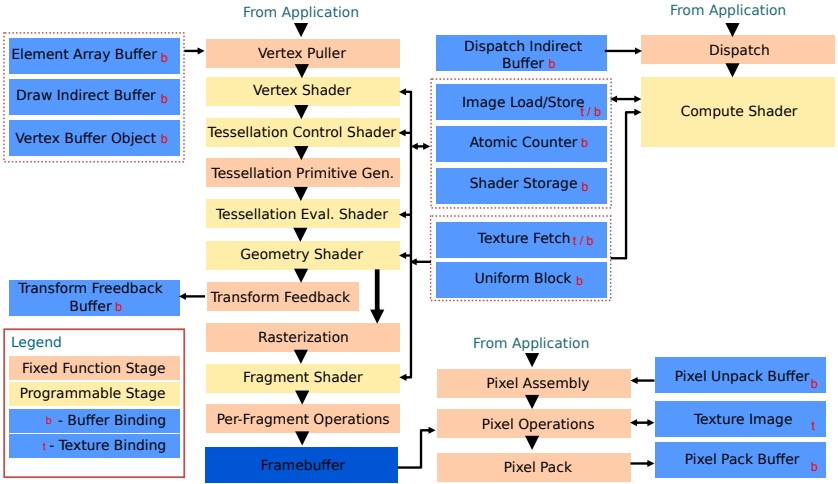
Atomic counter:

- Atomic increment/decrement.
- It can be useds as index into shader storage buffers.
- Indirect Draw

Shader Storage Buffer

- Random access memory for shaders
- Atomic operation
- Binding points

Atomic counters/Shader Storage



Atomic counters/Shader Storage - example

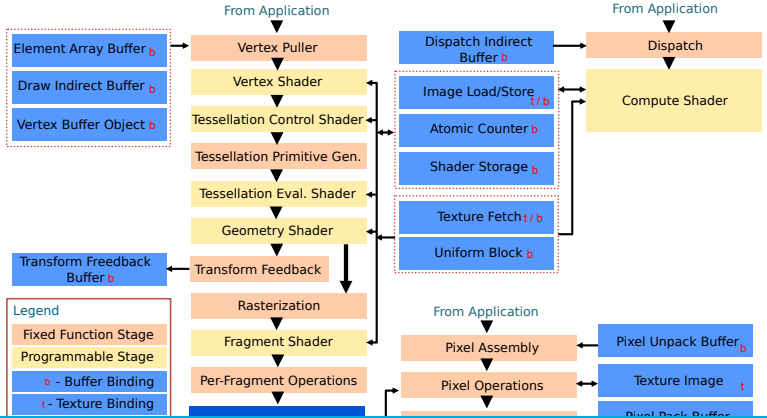
```
#version 430

in vec4 vPos;

layout(binding=1,offset=0)uniform atomic_uint counter;
layout(std430,binding=0)buffer Output{vec4 data[];};

layout(location=0)vec4 fColor;
//...
void main(){
  int W=atomicCounterIncrement(counter);//increment counter
  data[w*2+0]=ComputeColor(...);//compute color
  data[w*2+1]=vPos;//write position
  //...
}
```

Atomic counters/Shader Storage - example

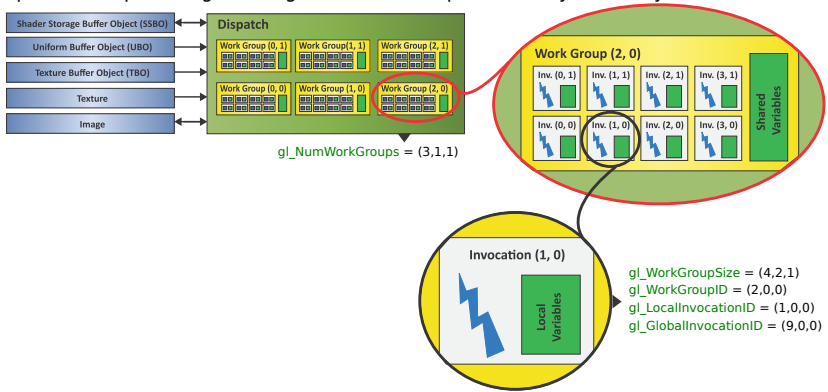
```
unsigned data[4]={0,0,0,0};
GLuint ACB;//identifier of a. counter
glCreateBuffers(1,&ACB);//create a. counter buffer
glNamedBufferData(ACB,sizeof(uint32_t)*4,data,GL_DYNAMIC_DRAW);
glBindBufferBase(GL_ATOMIC_COUNTER_BUFFER,1,ACB);//binding point 1

GLuint SSBO;//identifier of shader storage buffer
glCreateBuffers(1,&SSBO);//reserve identifier
glNamedBufferData(GL_SHADER_STORAGE_BUFFER,sizeof(float)*4*2*Max,
  NULL,GL_DYNAMIC_DRAW);
glClearNamedBufferData(SSBO,GL_R32F,GL_RED,GL_FLOAT,NULL);
glBindBufferRange(GL_SHADER_STORAGE_BUFFER,0,SSBO,0,
  sizeof(float)*4*2*Max);//binding point 0
```

Compute Shader

- Compute shader is located outside of common rendering pipeline.
- Compute shader can be used for general computing - GPGPU (other shader are in some way restricted).
- Compute shader share syntax with the rest of shader and can be used on different platforms.



OpenGL Compute Programming Model and Compute Memory Hierarchy



Compute Shader - example

```
#version 430

layout(binding=0)buffer Input{vec4 i[];};//input buffer
layout(binding=1)buffer Output{vec4 o[];};//output buffer
layout(local_size_x=32,local_size_y=1,local_size_z=1)in;//work-group size

uniform uint num;//number of elements in buffers

void main(){
  //gl_GlobalInvocationID = gl_WorkGroupID*gl_WorkGroupSize + gl_LocalInvocationID
  uint gid = gl_GlobalInvocationID.x;
  if(gid >= num)return;
  vec3 v = i[gid].xyz;
  o[gid] = vec4(normalize(v),length(v));
}
```

Compute Shader - example

```
uint32_t num=100;
GLuint CompBufferInput;
glCreateBuffers(1,&CompBufferInput);
glNamedBufferData(CompBufferInput,sizeof(float)*4*num,nullptr,GL_STATIC_DRAW);

GLuint CompBufferOutput;
glCreateBuffers(1,&CompBufferOutput);
glNamedBufferData(CompBufferOutput,sizeof(float)*4*num,nullptr,GL_DYNAMIC_COPY);

glBindBufferBase(GL_SHADER_STORAGE_BUFFER,0,CompBufferInput);
glBindBufferBase(GL_SHADER_STORAGE_BUFFER,1,CompBufferOutput);

glUseProgram(ComputeShader);
glUniformlui(glGetUniformLocation(ComputeShader,"num"),num);

uint32_t workGroupSize = 32;

glDispatchCompute(divRoundUp(num,workGroupSize),1,1);
```

Compute Shader - dispatch synchronization

- Synchronization needs to be done between two shaders thats operates on same data.
- Synchronization is performed using glMemoryBarrier(); command.

```
//This compute shader creates buffer with vertices.
glDispatchCompute(
    this->TileCount[(this->NumLevels-2)*2+0],
    this->TileCount[(this->NumLevels-2)*2+1],
    1);

//Wait for modification to SSBO
glMemoryBarrier(GL_SHADER_STORAGE_BARRIER_BIT);

//Draw generated vertices
glDrawArrays(...)
```

Compute Shader - thread synchronization

- Threads in work-group are execute in warp/wavefronts.
- We can synchronize threads that are part of work-group.
- We cannot synchronize threads that are part of different work-groups.
- Threads that are part of warp do not have to be synchronized.
- Synchronization is performed by barrier() command.
- A barrier command has to lie on program path that can be access by all threads in work-group.
- A barrier command stops every thread until all other threads in work-group reach it.

Example:

```
layout(std430,binding=0)readonly buffer SFData{float data[];};
shared float array[size];
void main(){
  //cooperative reading from global memory into local/shared memory.
  array[gl_LocalInvocationID.x]=data[gl_GlobalInvocationID.x];
  //wait for all threads
  barrier();
  //...
}
```

Queries

- Queries can be used to obtain some information about drawing.
- Number of rasterized samples.
- Number of generated primitives.
- Time of execution of commands.
- Conditional rendering - occlusion culling

```
void glGenQueries(GLsizei n,GLuint * ids);
```

```
void glBeginQuery(GLenum target,GLuint id);
```

```
void glEndQuery(GLenum target,GLuint id);
```

```
void glGetQueryiv(GLenum target,GLuint id,GLint * params);
```

Queries - time example

```
//init
GLuint QueryTime;//query
glGenQueries(1,&QueryTime);//nagenerujeme si ID query
GLuint QueryTimePassed=0;//cas v nanosekundach

//start query
glBeginQuery(GL_TIME_ELAPSED,QueryTime);

//commands
//glBindBuffer(...);
//glUseProgram(...);
//glDrawArrays(...);
//...

//end query
glEndQuery(GL_TIME_ELAPSED);//vypneme query

//get data from asychronous  query
glGetQueryObjectuiv(QueryTime,GL_QUERY_RESULT_NO_WAIT,&QueryTimePassed);
```

Queries - conditional rendering example

```
//init
GLuint QuerySample;//query
glGenQueries(1,&QueryTime);//nagenerujeme si ID query

//draw bounding boxes
//start query
glBeginQuery(GL_ANY_SAMPLES_PASSED,QuerySample);
//draw bounding boxes
//glBindBuffer(...);
//glUseProgram(...);
//glDrawArrays(...);
//...
glEndQuery(GL_ANY_SAMPLES_PASSED);//vypneme query

//conditional rendering
//start conditional rendering
glBeginConditionalRender(QuerySample,GL_QUERY_NO_WAIT);
//draw meshes with full details
//glBindBuffer(...);
//glUseProgram(...);
//glDrawArrays(...);
//...
glEndConditionalRender();//end conditional rendering
```

Synchronization objects, fences

- Synchronization between OpenGL commands.
- Host synchronization, client synchronization.

```
GLsync sync;//synchronization object
glDispatchCompute(...);
sync=glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE,0);
glClientWaitSync(sync,0,GL_TIMEOUT_IGNORED);//CPU waits
//...
glDeleteSync(sync);
```

```
GLsync sync;//synchronization object
glDispatchCompute(...);
sync=glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE,0);
glWaitSync(sync,0,GL_TIMEOUT_IGNORED);//GPU waits
//...
glDeleteSync(sync);
```

```
glFlush();
glFinish();
```

- OpenGL exposes commands for debugging.
- User can define custom messages, callbacks, ...
- Old way: glGetError
- OpenGL debugging is supported in debug OpenGL context

```
void glDebugMessageCallback(DEBUGPROC callback,void * userParam);
```

```
void glDebugMessageControl(GLenum,GLenum,GLenum,GLsizei,const GLuint*,...);
```

```
void glDebugMessageInsert(GLenum,GLenum,GLuint,GLenum,GLsizei,const char*);
```

Debugging - example

```
//create window using SDL2
SDL_Init(SDL_INIT_VIDEO|SDL_INIT_TIMER);//init. video
SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION,4);
SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION,3);
SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
  SDL_GL_CONTEXT_PROFILE_CORE);
//set debug context
SDL_GL_SetAttribute(SDL_GL_CONTEXT_FLAGS,SDL_GL_CONTEXT_DEBUG_FLAG);
SDL_CreateWindow("OpenGL s debug kontextem",SDL_WINDOWPOS_CENTERED,
  SDL_WINDOWPOS_CENTERED,1024,768,
  SDL_WINDOW_OPENGL|SDL_WINDOW_SHOWN|SDL_WINDOW_FULLSCREEN);
```

```
//custom callback
void MyDebug(GLenum Source,GLenum Type,GLuint Id,GLenum severity,
  GLsizei Length,const GLchar*Message,void*UserParam){
    std::cerr<<"MyDebug: "<<Message<<std::endl;
}

glEnable(GL_DEBUG_OUTPUT);//enable debugging
glDebugMessageCallback(MyDebug,NULL);//set callback
```

References

- http://www.opengl.org/sdk/docs/
- http://www.opengl.org/documentation/glsl/
- https://wiki.libsdl.org/FrontPage

Thank you for your attention! Questions?