MODERN OPENGL

APRIL 2012

Evolution of the OpenGL Pipe

GLSL, Shader Interfaces

Vertex Submission

Demo: Instancing, Layered FBOs

Demo: Tessellation Shaders

Transform Feedback

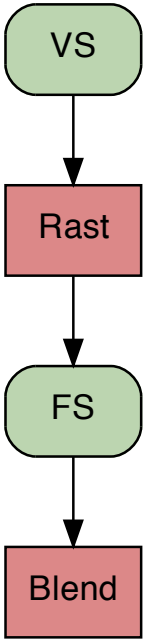
Textures, Buffer Objects

Misc 4.2 Features / Extensions

```glsl
// Vertex Shader
varying vec4 vPosition;
varying vec2 vTexCoord;
varying vec3 vNormal;
void main() {
    vPosition = gl_ModelViewMatrix * gl_Vertex;
    vNormal = gl_NormalMatrix * gl_Normal;
    vTexCoord = gl_MultiTexCoord0.st;
    gl_Position = ftransform();
}
```

```glsl
// Fragment Shader
uniform sampler2D Baboon;
uniform vec3 LightDir;
varying vec2 vTexCoord;
varying vec3 vNormal;
void main() {
    float df = dot(LightDir,vNormal);
    vec4 c = df * texture2D(Baboon, vTexCoord);
    gl_FragColor = vec4(c.rgb, 1);
}
```

```glsl
// Application
glTexCoordPointer(2, GL_FLOAT, sizeof(float)*2, &coords[0]);
glMatrixMode(GL_MODELVIEW);
```



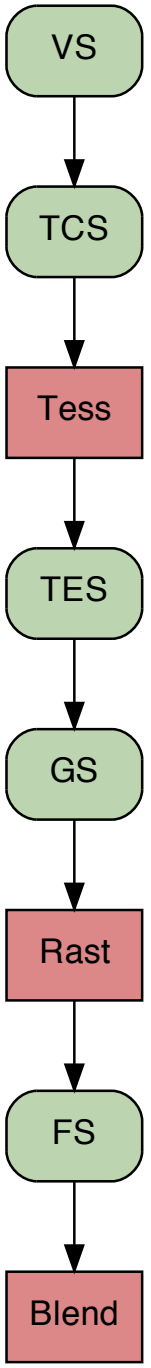
```glsl
// Vertex Shader
layout(location = 0) in vec4 Position;
layout(location = 1) in vec2 TexCoord;

out VSOut {
    vec4 Position;
    vec2 TexCoord;
} Out;

uniform mat4 ModelViewMatrix, MVP;

void main() {
    Out.Position = ModelViewMatrix * Position;
    Out.TexCoord = TexCoord;
    gl_Position = MVP * Out.Position;
}
```

```glsl
// Tessellation Control Shader
// ...
```

```glsl
// Tessellation Evaluation Shader
...
```

```glsl
// Fragment Shader
uniform sampler2D Baboon;
uniform vec3 LightDir = vec3(0,0,1);
out vec4 CrazyColorChannel;

in GSOut {
    vec3 Normal;
    vec2 TexCoord;
} In;

void main() {
    float df = dot(LightDir, In.Normal);
    vec4 c = df * texture(Baboon, In.TexCoord);
    CrazyColorChannel = vec4(c.rgb, 1);
}
```

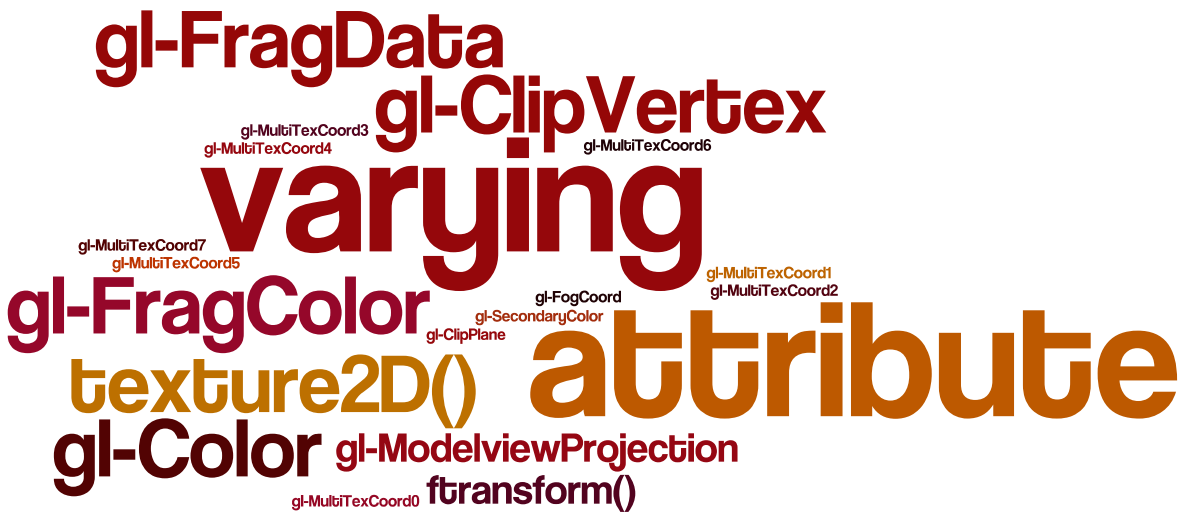
```glsl
// Geometry Shader
layout(triangles) in;
layout(triangle_strip, max_vertices = 3) out;

in TEOut {
    vec2 TexCoord;
    vec3 Normal;
    float Luminance[2];
} In[3];

out GSOut {
    vec3 Normal;
    vec2 TexCoord;
} Out;

subroutine vec3 NormFunc(vec3, vec3);
subroutine uniform NormFunc NormVar;

subroutine(NormFunc)
vec3 RobustNormal(vec3 A, vec3 B) {
    return normalize(cross(A, B));
}

subroutine(NormFunc)
vec3 CheapNormal(vec3 A, vec3 B) {
    return cross(A, B);
}

void main() {
    vec3 A = In[2].Position - In[0].Position;
    vec3 B = In[1].Position - In[0].Position;
    Out.Normal = NormVar(A, B);

    for (int j = 0; j < 3; j++) {
        Out.TexCoord = In[j].TexCoord;
        gl_Position = gl_in[j].gl_Position;
        EmitVertex();
    }
    EndPrimitive();
}
```



```
QGLFormat format;
format.setVersion(4,2);
format.setProfile(QGLFormat::CoreProfile);
QGLWidget *myWidget = new QGLWidget(format);
```

```
int attribs[] = {
    GLX_CONTEXT_MAJOR_VERSION_ARB, 4,
    GLX_CONTEXT_MINOR_VERSION_ARB, 2,
    GLX_CONTEXT_PROFILE_MASK_ARB, GLX_CONTEXT_CORE_PROFILE_BIT_ARB,
    NULL
};
GLXContext glc = glXCreateContextAttribs(diplay, config, NULL, True, attribs);
```

Core Profile


gl-FragData
gl-ClipVertex
gl-MultiTexCoord3
gl-MultiTexCoord4
gl-MultiTexCoord6
varying
gl-MultiTexCoord7
gl-MultiTexCoord5
gl-MultiTexCoord1
gl-MultiTexCoord2
gl-FragColor
gl-FogCoord
gl-SecondaryColor
gl-ClipPlane
attribute
texture2D()
gl-Color
gl-ModelviewProjection
gl-MultiTexCoord0
ftransform()

OLD


texelFetch()
texture()
gl-ViewportIndex
out
gl-TessLevelInner
textureQueryLod()
in
imageStore()
gl-ClipDistance[]
imageAtomicAdd()
layout
gl-Layer
imageLoad()
gl-TessLevelOuter
gl-InvocationID
bitfieldExtract()
gl-PrimitiveIDIn
textureSize()

NEW

Jurassic Vertices

```
glBegin(GL_TRIANGLES);
glColor4f(1, 0, 0, 0
glVertex3f(0, 1,
glVertex3f(1, 1
glVertex3f(1,
glEnd(GL_TRIA
```

```
glVertexPoint
glColorPointe
glNormalPointe
```

```
glNewList
glCallList
```

```
GL_QUAD_STRIP, GL_QUADS, GL_POLYGON
```

Modern Vertices

```
glVertexAttrib3d
glVertexAttrib4i
glVertexAttribI4i
glVertexAttribL2d
etc...
```

```
glVertexAttribPointer
glVertexAttribIPointer
glVertexAttribLPointer
```

```
GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_TRIANGLES
GL_LINE_STRIP_ADJACENCY, GL_LINES_ADJACENCY, GL_TRIANGLE_STRIP_ADJACENCY, GL_TRIANGLES_ADJACENCY
GL_PATCHES
```

Vertex Array Objects

```
const GLuint PositionSlot = 0;
const GLuint NormalSlot = 1;

GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);

glEnableVertexAttribArray(PositionSlot);
glEnableVertexAttribArray(NormalSlot);

glBindBuffer(GL_ARRAY_BUFFER, positionsVbo);
glVertexAttribPointer(PositionSlot, 3, GL_FLOAT, GL_FALSE,
                      sizeof(float)*3, 0);

glBindBuffer(GL_ARRAY_BUFFER, normalsVbo);
glVertexAttribPointer(NormalSlot, 3, GL_FLOAT, GL_FALSE,
                      sizeof(float)*3, 0);
```

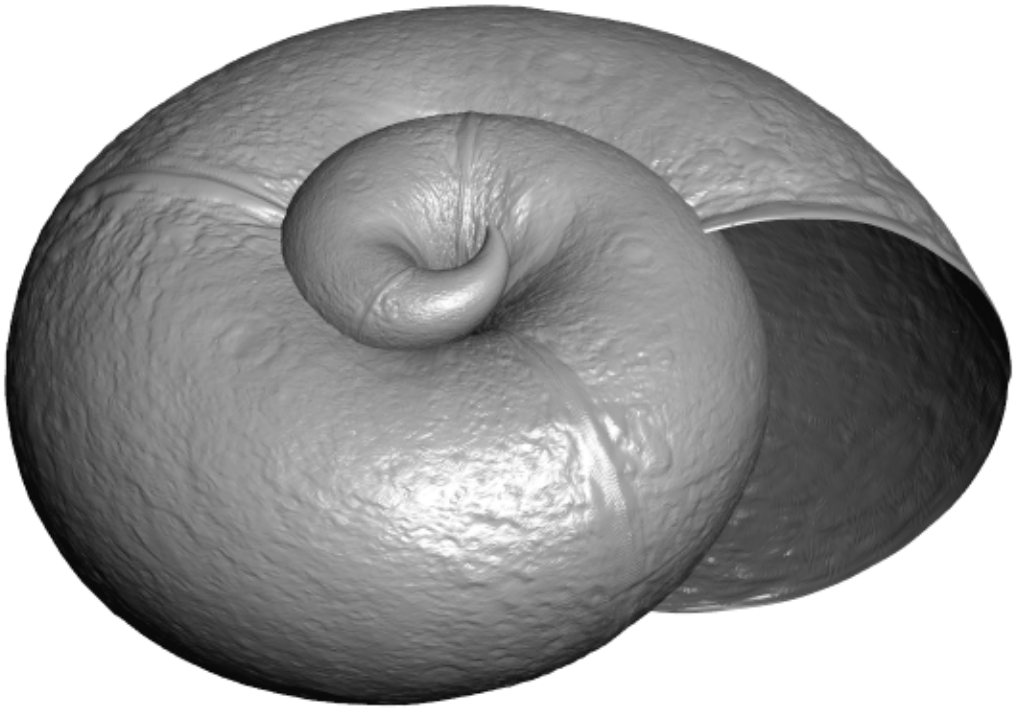
Buffer Objects

All Buffer Targets glBufferData glBufferSubData glMapBufferRange glCopyBufferSubData	**GL_PIXEL_PACK_BUFFER** glTexImage* glTexSubImage* glDrawPixels
GL_PIXEL_UNPACK_BUFFER glGetTexImage* glGetTexSubImage* glReadPixels	**GL_ARRAY_BUFFER** glVertexAttrib*
GL_ELEMENT_ARRAY_BUFFER glDrawElements (etc)	**GL_DRAW_INDIRECT_BUFFER** glDrawArraysIndirect glDrawElementsIndirect
GL_UNIFORM_BUFFER glUniformBlockBinding	

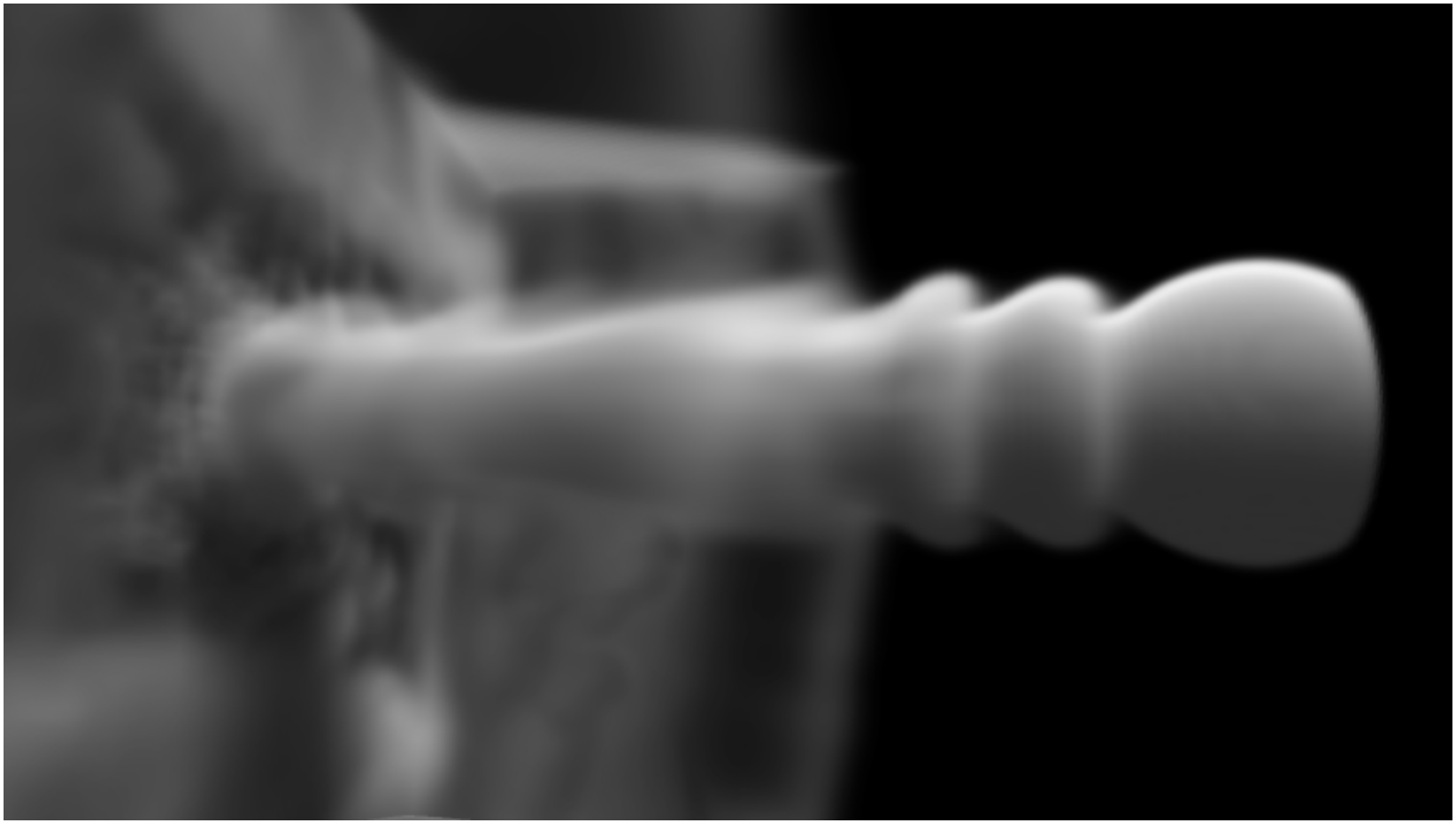
Indirect Drawing

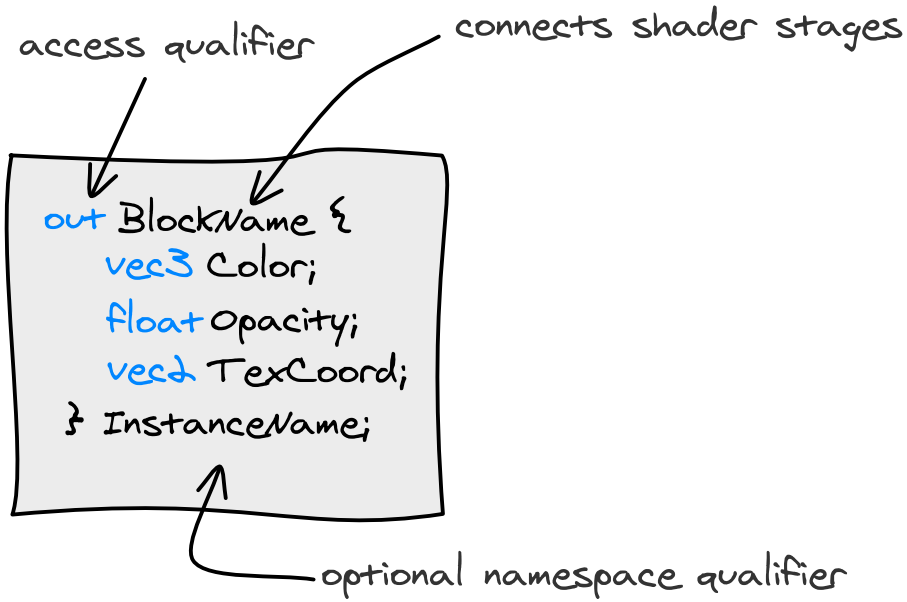
```
GLuint mydrawcall[] = {
    62, /* count */
    12, /* primcount */
    0,  /* first */
    0,  /* baseInstance */
};

// Get parameters from CPU memory:
glDrawArraysInstancedBaseInstance(GL_TRIANGLES, 62, 12, 0, 0);
glDrawArraysIndirect(GL_TRIANGLES, mydrawcall);

// Get parameters from GPU memory:
GLuint bufObj;
glGenBuffers(1, &bufObj);
glBindBuffer(GL_DRAW_INDIRECT_BUFFER, bufObj);
glBufferData(GL_DRAW_INDIRECT_BUFFER, sizeof(mydrawcall), mydrawcall, GL_STATIC_DRAW);
glDrawArraysIndirect(GL_TRIANGLES, 0);

// Generate parameters from OpenCL:
glGenBuffers(1, &bufObj);
glBindBuffer(GL_DRAW_INDIRECT_BUFFER, bufObj);
glBufferData(GL_DRAW_INDIRECT_BUFFER, sizeof(mydrawcall), NULL, GL_STATIC_DRAW);
clCreateFromGLBuffer(context, CL_MEM_READ_WRITE, bufObj, &err);
```

DEMO



DEMO






```
-- Vertex Shader

out MyBlock {
    vec3 Position;
    vec3 Color[2];
    float Opacity;
} Out;

-- Geometry Shader

in MyBlock {
    vec3 Position;
    vec3 Color[2];
    float Opacity;
} In[];
```

```
-- Vertex Shader

// Built-ins:
out gl_PerVertex {
    vec4 gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[];
};

// User-defined:
in MyBlock {
    float w;
} In;

void main()
{
    gl_Position = vec4(1, 0, 0, In.w);
}
```

```
GLuint i = glGetAttribLocation(program, "MyBlock.w");
```

Binding Vertex Attributes

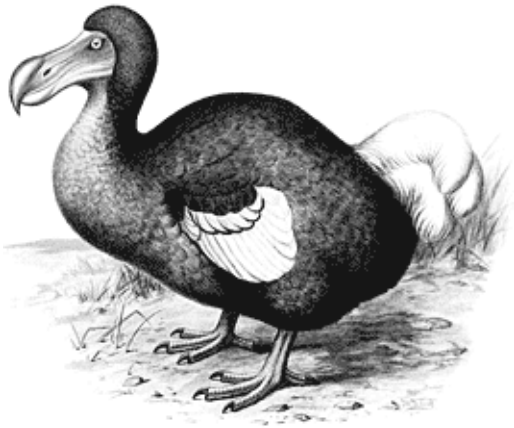
```
// Worst: let the compiler decide
GLuint foo = glGetAttribLocation(program, "MyBlock.w");
```

```
// Better: Specify in application code
GLuint foo = 3;

glCompileShader(vsHandle);
glAttachShader(programHandle, vsHandle);
glBindAttribLocation(programHandle, foo, "MyBlock.w");
glLinkProgram(programHandle);
```

```
// Best: Declare in GLSL
in MyBlock {
    layout(location = 3) vec3 w;
}
```

```
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glVertexAttribPointer(foo, 1, GL_FLOAT, GL_FALSE, stride, 0);
glEnableVertexAttribArray(foo);
```

gl_FragColor
gl_FragData[*n*]



Binding Fragment Outputs

```
// Let the compiler decide
GLuint colorNumber = glGetFragDataLocation(program, "MyColorVariable");
```

```
// Specify in application code
GLuint colorNumber = 3;
glBindFragDataLocation(programHandle, colorNumber, "MyColorVariable");
```

```
// Declare in GLSL
layout(location = 3) out vec4 factor;
```

```
 // Beware, a level of indirection!
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, myFbo);

GLenum buffers[] = {GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1};
glDrawBuffers(2, &buffers[0]);
```

Subroutines

```
-- Vertex Shader

subroutine vec3 IlluminationFunc(vec3 N, vec3 L);

subroutine(IlluminationFunc)
vec3 diffuse(vec3 N, vec3 L)
{
    return max(0, dot(N, L));
}

subroutine(IlluminationFunc)
vec3 specular(vec3 N, vec3 L)
{
    vec3 E = vec3(0, 0, 1);
    vec3 H = normalize(L + E);
    return pow(dot(N, H), Shininess);
}

uniform float Shininess = 1.0;
subroutine uniform IlluminationFunc IlluminationVar;

out vec4 vColor;
void main()
{
    vec3 n = vec3(0, 0, 1);
    vec3 p = vec3(3, 1, 4);
    vec3 c = IlluminationVar(n, p);
    vColor = vec4(c, 1);
}

-- Geometry Shader

// normal uniforms are scoped to the program object:
uniform float Shininess = 1.0;

// subroutines are scoped to the shader stage:
subroutine vec3 IlluminationFunc(float foo);
subroutine uniform IlluminationFunc IlluminationVar;
```

```
GLuint prog;
glGetIntegerv(GL_CURRENT_PROGRAM, &prog);

GLenum vs = GL_VERTEX_SHADER;

GLuint illum = glGetSubroutineUniformLocation(prog, vs,
                                              "IlluminationVar");

GLuint diffuse = glGetSubroutineIndex(prog, vs, "diffuse");
GLuint specular = glGetSubroutineIndex(prog, vs, "specular");

// This sets per-context state:
GLuint indices[MAX_SUBROUTINE_VARIABLES];
indices[illum] = diffuse;
glUniformSubroutinesuiv(GL_VERTEX_SHADER, 1, indices);

// This sets per-program state:
GLuint shiny = glGetUniformLocation(prog, "Shininess");
glUniform1f(prog, shiny, 1.0);
```

```
static GLuint LoadProgram(const char* vsSource,
                          const char* gsSource,
                          const char* fsSource)
{
    GLuint programHandle = glCreateProgram();
    GLuint vsHandle = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vsHandle, 1, &vsSource, 0);
    glCompileShader(vsHandle);
    glAttachShader(programHandle, vsHandle);
    if (gsSource != NULL) {
        GLuint gsHandle = glCreateShader(GL_GEOMETRY_SHADER);
        glShaderSource(gsHandle, 1, &gsSource, 0);
        glCompileShader(gsHandle);
        glAttachShader(programHandle, gsHandle);
    }
    GLuint fsHandle = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fsHandle, 1, &fsSource, 0);
    glCompileShader(fsHandle);
    glAttachShader(programHandle, fsHandle);
    glLinkProgram(programHandle);
    GLint linkSuccess;
    glGetProgramiv(programHandle, GL_LINK_STATUS, &linkSuccess);
    if (!linkSuccess) {
        GLchar spew[256];
        glGetProgramInfoLog(programHandle, sizeof(spew), 0, spew);
        fprintf(stderr, "Can't link shaders:\n%s", spew);
        return 0;
    }
    glUseProgram(programHandle);
    return programHandle;
}
```

OLD SCHOOL

```
static GLuint LoadProgram(const char* vsSource, const char* gsSource, const char* fsSource)
{
    GLuint programHandle = glCreateProgram();
    if (vsSource != NULL) {
        GLuint vsHandle = glCreateShader(GL_VERTEX_SHADER);
        glShaderSource(vsHandle, 1, &vsSource, 0);
        glCompileShader(vsHandle);
        glAttachShader(programHandle, vsHandle);
    }
    if (gsSource != NULL) {
        GLuint gsHandle = glCreateShader(GL_GEOMETRY_SHADER);
        glShaderSource(gsHandle, 1, &gsSource, 0);
        glCompileShader(gsHandle);
        glAttachShader(programHandle, gsHandle);
    }
    if (fsSource != NULL) {
        GLuint fsHandle = glCreateShader(GL_FRAGMENT_SHADER);
        glShaderSource(fsHandle, 1, &fsSource, 0);
        glCompileShader(fsHandle);
        glAttachShader(programHandle, fsHandle);
    }
    glProgramParameteri(programHandle, GL_PROGRAM_SEPARABLE, GL_TRUE);
    glLinkProgram(programHandle);
    GLint linkSuccess;
    glGetProgramiv(programHandle, GL_LINK_STATUS, &linkSuccess);
    if (!linkSuccess) {
        GLchar spew[256];
        glGetProgramInfoLog(programHandle, sizeof(spew), 0, spew);
        fprintf(stderr, "Can't link shaders:\n%s", spew);
        return 0;
    }
    glUseProgram(programHandle);
    return programHandle;
}
```

```
static GLuint LoadPipeline(
        const char* vsSource,
        const char* gsSource,
        const char* fsSource)
{
    GLuint vsProgram = glCreateShaderProgramv(GL_VERTEX_SHADER, 1, &vsSource);
    GLuint gsProgram = glCreateShaderProgramv(GL_GEOMETRY_SHADER, 1, &gsSource);
    GLuint fsProgram = glCreateShaderProgramv(GL_FRAGMENT_SHADER, 1, &fsSource);

    GLuint pipeline;
    glGenProgramPipelines(1, &pipeline);
    glBindProgramPipeline(pipeline);

    glUseProgramStages(pipeline, GL_VERTEX_SHADER_BIT, vsProgram);
    glUseProgramStages(pipeline, GL_GEOMETRY_SHADER_BIT, gsProgram);
    glUseProgramStages(pipeline, GL_FRAGMENT_SHADER_BIT, fsProgram);

    // glUniform* now heed the "active" shader program rather than glUseProgram
    glActiveShaderProgram(pipeline, vsProgram);
    glUniform1f(fooLocation, 1.0f);

    return pipeline;
}
```

Separable Programs

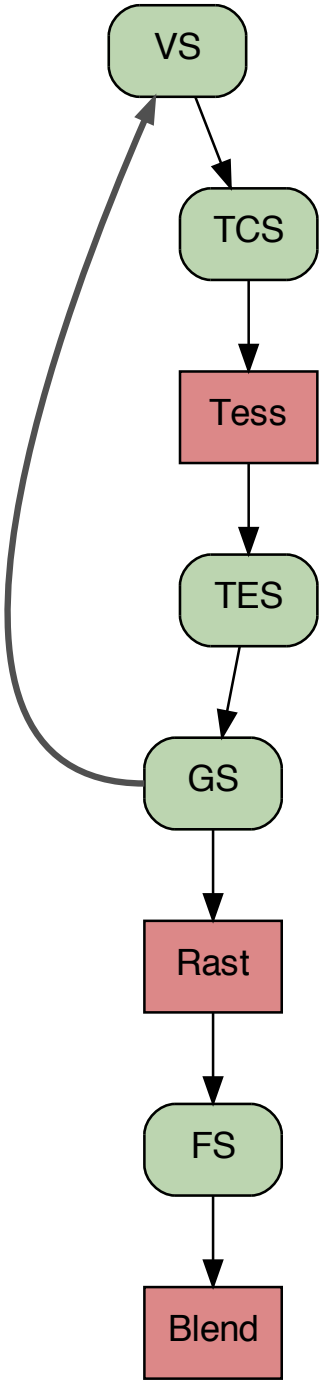
```
...

glProgramParameteri(programHandle, GL_PROGRAM_BINARY_RETRIEVABLE_HINT, GL_TRUE);
glLinkProgram(programHandle);

GLuint bufSize;
glGetProgramiv(programHandle, GL_PROGRAM_BINARY_LENGTH, &bufSize);

std::vector buffer(bufSize);

GLenum binaryFormat;
glGetProgramBinary(programHandle, bufSize, NULL, &binaryFormat, &buffer[0]);
```

```
// use a cached program on subsequent runs:
glProgramBinary(programHandle, binaryFormat, &buffer[0], bufSize);
```

Binaries!

```
// This goes after glCompileShader but before glLinkProgram...
const char* varyings[4] = { "vPosition", "gl_NextBuffer", "vBirthTime", "vVelocity" };
glTransformFeedbackVaryings(programHandle, 4, varyings, GL_INTERLEAVED_ATTRIBS);

// Create VBO for input on even frames and output on odd frames:
glGenBuffers(1, &BufferA);
glBindBuffer(GL_ARRAY_BUFFER, BufferA);
glBufferData(GL_ARRAY_BUFFER, sizeof(seed_data), &seed_data[0], GL_STREAM_DRAW);

// Create VBO for output on even frames and input on odd frames:
glGenBuffers(1, &BufferB);
glBindBuffer(GL_ARRAY_BUFFER, BufferB);
glBufferData(GL_ARRAY_BUFFER, sizeof(seed_data), 0, GL_STREAM_DRAW);

// Create a transform feedback object:
GLuint Feedback = 0;
glGenTransformFeedbacks(1, &Feedback);
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, Feedback);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, BufferA);
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, 0);
```

```
glEnable(GL_RASTERIZER_DISCARD);
glBindBuffer(GL_ARRAY_BUFFER, BufferA);
glVertexAttribPointer(...);
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, TransformFeedback);
glBeginTransformFeedback(GL_POINTS);
glDrawArrays(GL_POINTS, 0, inCount);
glEndTransformFeedback();
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, 0);

swap(BufferA, BufferB);

glDisable(GL_RASTERIZER_DISCARD);
glBindBuffer(GL_ARRAY_BUFFER, BufferA);
glVertexAttribPointer(...);
glDrawTransformFeedback(GL_POINTS, TransformFeedback); // similar to glDrawArrays
```

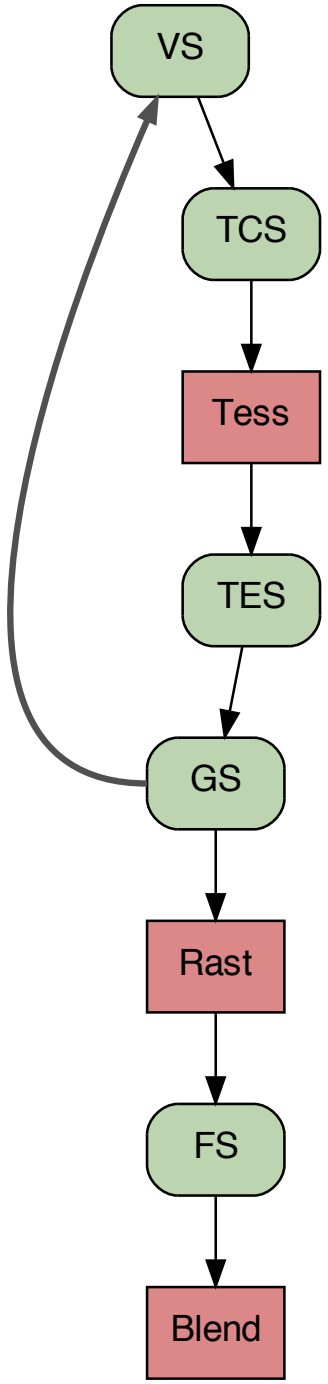


New Transform Feedback

```
// Assign streams in geometry shader
(layout out = 0) out vec4 vPosition;
(layout out = 1) out vec4 vBirthTime;
(layout out = 1) out vec4 vVelocity;
...
EmitStreamVertex(0);
EmitStreamPrimitive(0);
```

```
// Assign varyings to "record" during initialization
const char* varyings[4] = { "vBirthTime", "vVelocity" };
glTransformFeedbackVaryings(programHandle, 2, varyings,
                            GL_INTERLEAVED_ATTRIBS);
```

```
// This time, don't discard rasterization
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, TransformFeedback);
glBeginTransformFeedback(GL_POINTS);
glDrawArrays(GL_POINTS, offset0, count0);
glPauseTransformFeedback();
glDrawArrays(GL_POINTS, offset1, count1);
glResumeTransformFeedback();
glDrawArrays(GL_POINTS, offset2, count2);
glEndTransformFeedback();
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, 0);
```



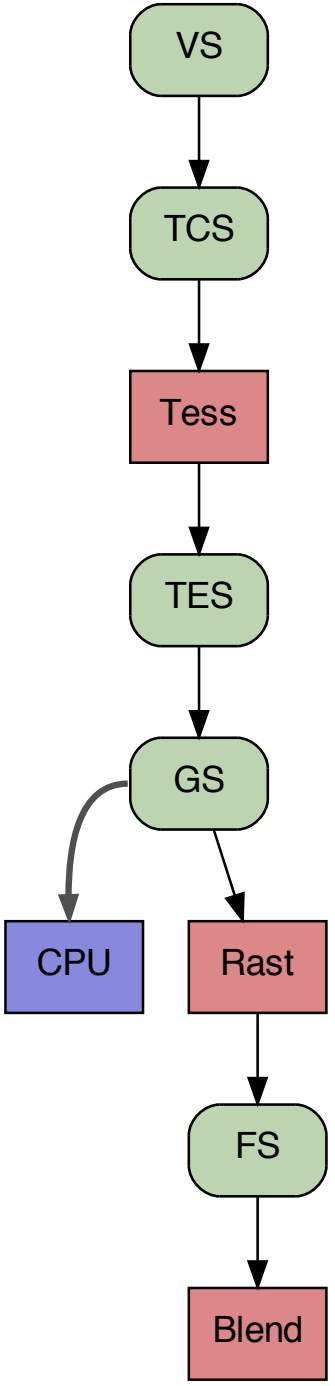
Multiple Streams / Pause / Resume

```
GLuint RecordBuffer, DrawBuffer; // VBOs
GLuint Feedback;                 // TFO

// ...

glGenTransformFeedbacks(1, &Feedback);
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, Feedback);
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, RecordBuffer);
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, 0);
```

```
glBindBuffer(GL_ARRAY_BUFFER, DrawBuffer);
glVertexAttribPointer(...);
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, TransformFeedback);
glBeginTransformFeedback(GL_POINTS);
glDrawArrays(GL_POINTS, offset, count);
glEndTransformFeedback();
glBindTransformFeedback(GL_TRANSFORM_FEEDBACK, 0);
```

```
glBindBuffer(GL_ARRAY_BUFFER, RecordBuffer);
void* rawdata = glMapBuffer( GL_ARRAY_BUFFER, GL_READ_ONLY);
// ...do stuff here...
glUnmapBuffer(rawData);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Send back to CPU



Texture Formats

```
// LUMINANCE and LUMINANCE_ALPHA et al are gone!
GLenum internalFormat = GL_RGB;
GLenum format = GL_RGB;
GLenum type = GL_UNSIGNED_BYTE;
glTexImage2D(GL_TEXTURE_2D, 0, internalFormat, width, height, 0, format, type, data);
```

INTERNAL FORMATS

DEPTH_COMPONENT	DEPTH_STENCIL	RED	RG	RGB	RGBA			
R8	R8_SNORM	R16	R16_SNORM	RG8	RG8_SNORM	RG16	RG16_SNORM	R3_G3_B2
RGB4	RGB5	RGB8	RGB8_SNORM	RGB10	RGB12	RGB16	RGB16_SNORM	RGBA2
RGBA4	RGB5_A1	RGBA8	RGBA8_SNORM	RGB10_A2	RGB10_A2UI	RGBA12	RGBA16	RGBA16_SNORM
SRGB8	SRGB8_ALPHA8	RGBA	R16F	RG16F	RGB16F	RGBA16F	R32F	RG32F
RGB32F	RGBA32F	R11F_G11F_B10F	RGB9_E5	R8I	R8UI	R16I	R16UI	R32I
R32UI	RG8I	RG8UI	RG16I	RG16UI	RG32I	RG32UI	RGB8I	RGB8UI
RGB16I	RGB16UI	RGB32I	RGB32UI	RGBA8I	RGBA8UI	RGBA16I	RGBA16UI	RGBA32I
RGBA32UI								

FORMATS

DEPTH_COMPONENT	DEPTH_STENCIL	RED	RG	RGB	RGBA	
STENCIL_INDEX	GREEN	BLUE	BGR	BGRA	RED_INTEGER	
GREEN_INTEGER	BLUE_INTEGER	RG_INTEGER	RGB_INTEGER	RGBA_INTEGER	BGR_INTEGER	
BGRA_INTEGER						

TYPES

UNSIGNED_BYTE	BYTE	UNSIGNED_SHORT	SHORT	
UNSIGNED_INT	INT	HALF_FLOAT	FLOAT	
UNSIGNED_SHORT_4_4_4_4	UNSIGNED_INT_8_8_8_8	UNSIGNED_INT_8_8_8_8_REV	UNSIGNED_INT_10_10_10_2	etc...

Compressed Textures

```
#define GL_COMPRESSED_RED_RGTC1           0x8DBB // Also known as: DXT_BC5, LATC, RGTC, 3Dc, ATI2
#define GL_COMPRESSED_SIGNED_RED_RGTC1    0x8DBC
#define GL_COMPRESSED_RG_RGTC2            0x8DBD
#define GL_COMPRESSED_SIGNED_RG_RGTC2     0x8DBE

#define GL_COMPRESSED_RGBA_BPTC_UNORM         0x8E8C // Also known as: DXT_BC7
#define GL_COMPRESSED_SRGB_ALPHA_BPTC_UNORM   0x8E8D
#define GL_COMPRESSED_RGB_BPTC_SIGNED_FLOAT   0x8E8E
#define GL_COMPRESSED_RGB_BPTC_UNSIGNED_FLOAT 0x8E8F

glCompressedTexImage3D (enum target, int level, enum internalformat, sizei width, sizei height,
                        sizei depth, int border, sizei imageSize, const void *data)

glCompressedTexImage2D (enum target, int level, enum internalformat, sizei width, sizei height,
                        int border, sizei imageSize, const void *data)

glCompressedTexImage1D (enum target, int level, enum internalformat, sizei width, int border,
                        sizei imageSize, const void *data)

glCompressedTexSubImage3D (enum target, int level, int xoffset, int yoffset, int zoffset,
                           sizei width, sizei height, sizei depth, enum format, sizei imageSize,
                           const void *data)

glCompressedTexSubImage2D (enum target, int level, int xoffset, int yoffset, sizei width,
                           sizei height, enum format, sizei imageSize, const void *data)

glCompressedTexSubImage1D (enum target, int level, int xoffset, sizei width, enum format,
                           sizei imageSize, const void *data)
```

Texture Buffers

```
GLuint bufObj;
glGenBuffers(1, &bufObj);
glBindBuffer(GL_TEXTURE_BUFFER, bufObj);
glBufferData(GL_TEXTURE_BUFFER, sizeof(data), data, GL_STREAM_DRAW);

GLenum sizedFormat = GL_RGBA32F;
glTexBuffer(GL_TEXTURE_BUFFER, sizedFormat, bufObj);
```

```
uniform samplerBuffer Foo;
...
int coord = ...;
vec4 color = texelFetch(Foo, coord);
```

```
glBindBuffer(GL_ARRAY_BUFFER, vbo);   // source
glBindBuffer(GL_TEXTURE_BUFFER, tbo); // destination
glBufferData(GL_TEXTURE_BUFFER, 16384, 0, GL_STREAM); // give it a size

GLintptr readoffset = 0, writeoffset = 0;
glCopyBufferSubData(GL_ARRAY_BUFFER, GL_TEXTURE_BUFFER,
                    readoffset, writeoffset, 16384);
```

Pixel Buffers

```
GLuint bufObj, texObj;

glGenBuffers(1, &bufObj);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER, bufObj);
glBufferData(GL_PIXEL_UNPACK_BUFFER, sizeof(data), data, GL_STREAM_DRAW);

glGenTextures(1, &texObj);
glBindTexture(GL_TEXTURE_2D, texObj);
glTexImage2D(..., NULL);
```

```
// Render with PBO 'A' while uploading PBO 'B'
glBindTexture(GL_TEXTURE_2D, texObj);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pboA);
glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, w, h, GL_RGBA, GL_UNSIGNED_BYTE, 0);

glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pboB);
glBufferData(GL_PIXEL_UNPACK_BUFFER, byteCount, 0, GL_STREAM_DRAW);

GLubyte* data = glMapBufferRange(GL_PIXEL_UNPACK_BUFFER, 0, byteCount, GL_MAP_WRITE_BIT);
// write stuff to 'data' here...
glUnmapBuffer(GL_PIXEL_UNPACK_BUFFER); // see also: glFlushMappedBufferRange

glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
std::swap(pboA, pboB);

// render here...
```

Direct State Access

```
uniform vec3 foo = vec3(1, 1, 2);
uniform vec3 bar = vec3(3, 5, 8);
```

```
// Old way
glUseProgram(prog1);
glGetUniformLocation("foo", &loc1);
glUniform3f(loc1, 3.14, 2.72, 1.62);
glUseProgram(prog2);
glGetUniformLocation("bar", &loc2);
glUniform3f(loc2, 3.14, 2.72, 1.62);

// New way
glProgramUniform3f(prog1, loc1, 3.14, 2.72, 1.62);
glProgramUniform3f(prog2, loc2, 3.14, 2.72, 1.62);
```

also check out EXT_direct_state_access

Conditional Rendering

```
GLuint query;
glGenQueries(1, &query);
...
glColorMaski(0, GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glDepthMask(GL_FALSE);
glBeginQuery(GL_ANY_SAMPLES_PASSED, query);
// ...render bounding box...
glEndQuery(...);
glEndQuery(GL_ANY_SAMPLES_PASSED);
glColorMaski(0, GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glDepthMask(GL_TRUE);

// ...render various stuff while waiting for results...

glBeginConditionalRender(query, GL_QUERY_WAIT);
// ...render full geometry...
glEndConditionalRender();
```

GL_QUERY_NO_WAIT
GL_QUERY_BY_REGION_WAIT, GL_QUERY_BY_REGION_NO_WAIT

Image Load / Store

```
uniform image2D alphaImage;
uniform iimage1D betaImage;
...
vec4 color = ...;
ivec2 coord = ...;
imageStore(alphaImage, coord, color);
...
color = imageLoad(alphaImage, coord);
...
int i = ...; // 1D coordinate
int foo = imageAtomicAdd(betaImage, i, 17)
```

```
GLuint imageLoc = glGetUniformLocation(prog, "alphaImage");
glUniform1i(imageLoc, 3); // must be < GL_MAX_IMAGE_UNITS

glBindImageTexture(3, texObj, miplevel,
                   GL_FALSE, 0, // <-- for layered textures
                   GL_READ_WRITE, GL_RGBA8);
```

see also: coherent volatile restrict readonly writeonly memoryBarrier()

Stuff I missed...

Atomic Counters
GL_ARB_debug_output
Viewport Arrays
Dual Source Blending

Bindless Graphics [nv prezo]

NV_bindless_texture
NV_shader_buffer_load
NV_vertex_buffer_unified_memory

```
uniform sampler2D* foo;
```

```
glMakeTextureHandleResidentNV(...);
```