Intro to OpenGL

CS4620 Lecture 14

Guest Instructor: Nicolas Savva

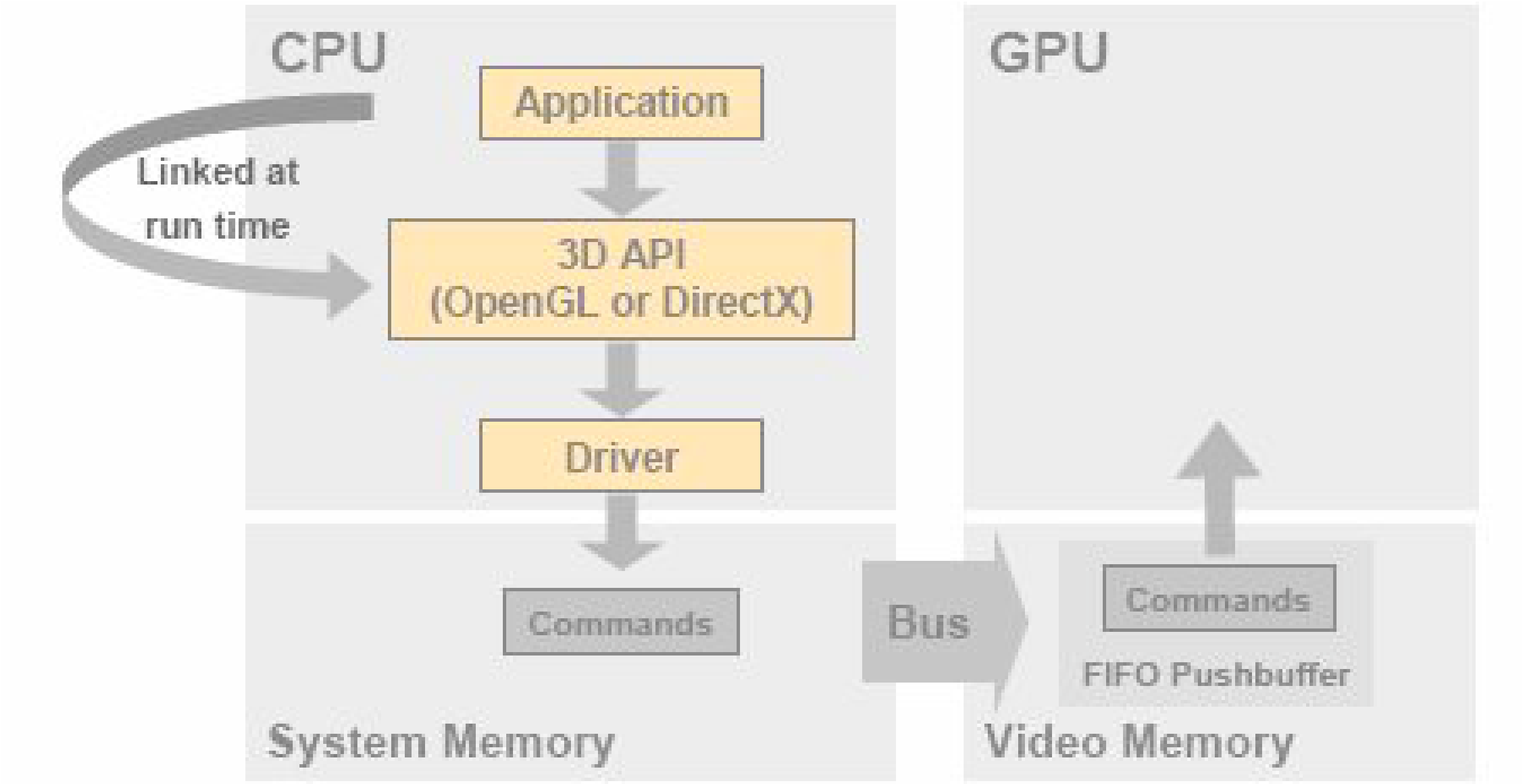
What is OpenGL?

- **Open G**raphics **L**ibrary
- A low level API for 2D/3D rendering with the Graphics Hardware (GPU)
- Cross-platform (Windows, OS X, Linux, iOS, Android, ...)

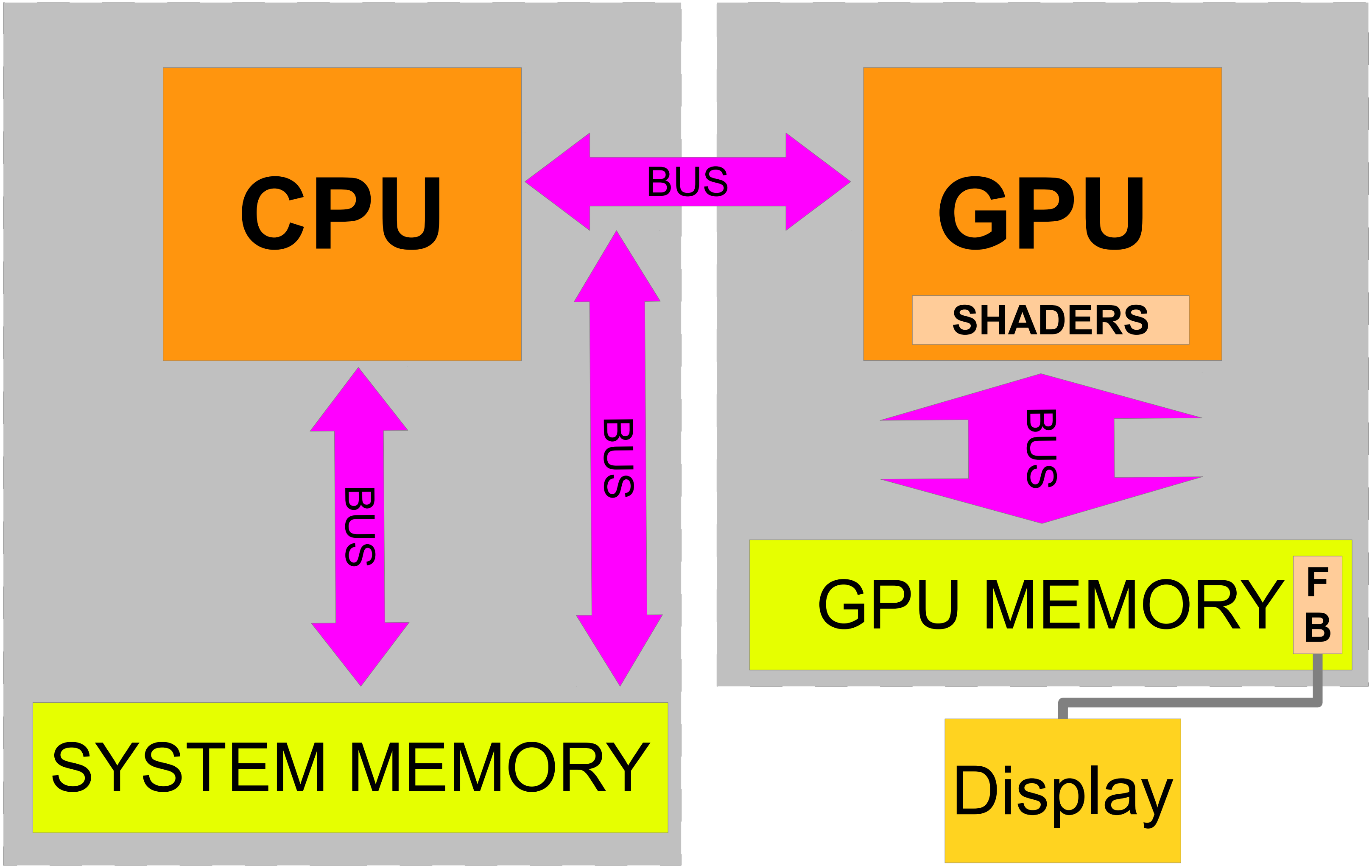
- Developed by SGI in 1992
  - 2014: OpenGL 4.5
  - 2008: OpenGL 3.0
  - 2006: OpenGL 2.1
- Main alternative: DirectX/3D

How does it fit in?

- Tap massive power of GPU hardware to render images
- Use GPU without caring about the exact details of the hardware



CPU and GPU memory



What OpenGL does for us

- Controls GPU
- Lets user specify resources...:
  - Geometry (vertices and primitives)
  - Textures
  - Shaders (programmable pieces of rendering pipeline)
  - Etc.
- ...and use them:
  - Rasterize and draw geometry

How we will use OpenGL

- OpenGL version
  - We use 2.x-3.x (plus extensions)
  - Code avoids older, deprecated parts of OpenGL standard
- LWJGL
  - Lightweight Java Game Library
  - Java bindings for OpenGL API
- CS 4620/4621 Framework
  - Simplifies creating and using OpenGL resources

LWJGL

- OpenGL originally written for C.
- LWJGL contains OpenGL binding for Java www.lwjgl.org/

- Gives Java interface to C OpenGL commands
- Manages framebuffer

(framebuffer: a buffer that holds the image that is displayed on the monitor)

LWJGL
Lightweight Java Game Library

MainGame

- A window which can display GameScreens
- Initializes OpenGL context
- Forwards keyboard and mouse events to the event dispatcher

- Usage
  - Inherit from MainGame and implement methods
  - Create instance and call run method

MainGame

```
@Override
protected void buildScreenList() {
   // Create objects inherited from GameScreen and
   // initialize screenList attribute
}

@Override
protected void fullInitialize() {
   // Code Executed Before Window Is Created
}

@Override
protected void fullLoad() {
   // Code Executed With An Active OpenGL Context
}
```

GameScreen

- Can display images created by OpenGL
- OpenGL “context”
  - Stores OpenGL state (geometry, buffers, etc.)

- Usage:
  - Inherit from class and implement methods
  - Create instance in MainGame.buildScreenList

GameScreen

```
@Override
public void update(GameTime gameTime) {
  // Animation: Update position of scene objects, camera
}

@Override
public void draw(GameTime gameTime) {
  // Drawing: Use LWJGL to draw to the screen
}

@Override
public void onEntry(GameTime gameTime) {
  // Initialization code
}

@Override
public void onExit(GameTime gameTime) {
  // Destruction, free allocated resources here
}
```

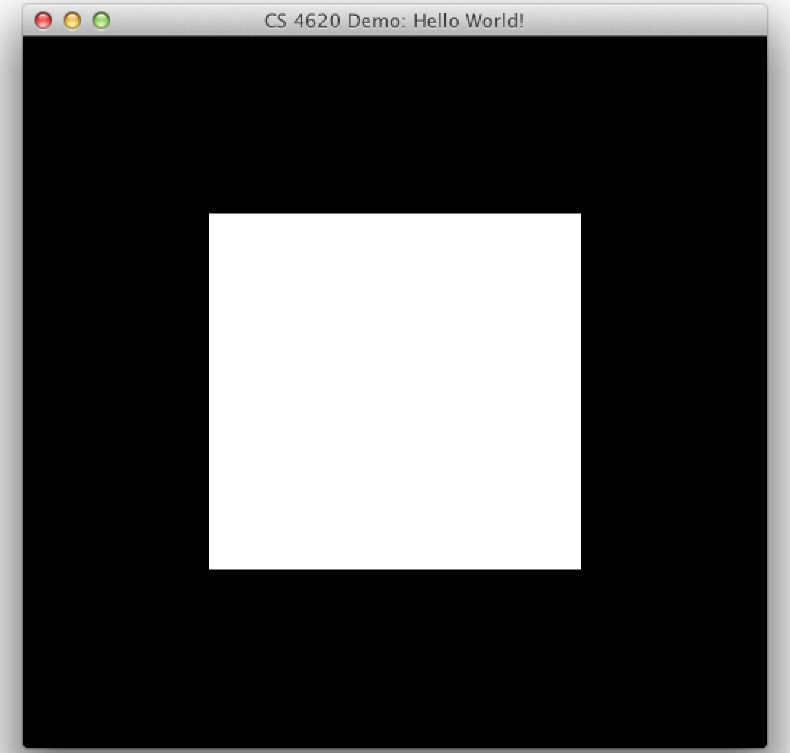
Events

- MainGame can trigger certain events
  - Something happens (e.g. user resizes window)
  - MainGame forwards event to the event dispatcher
    - KeyboardEventDispatcher
    - MouseEventDispatcher
  - Objects interested in the event can sign up as listeners
    - e.g. KeyboardEventDispatcher.OnKeyPressed.add(…)

- These events let us interact with OpenGL

OpenGL Commands and Resources

Demo: Hello World!



Example: Hello World's draw()

```
@Override
public void draw(GameTime gameTime) {
    GL11.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    GL11.glClear(GL11.GL_COLOR_BUFFER_BIT);

    program.use();

            GLUniform.setST(program.getUniform("VP"),
            new Matrix4(), false);
            GLUniform.set(program.getUniform("uGridColor"),
            new Vector4(1, 1, 1, 1));

            vb.useAsAttrib(program.getAttribute("vPos"));
            ib.bind();
            GL11.glDrawElements(GL11.GL_TRIANGLES, indexCount,
            GLType.UnsignedInt, 0);
            ib.unbind();

    GLProgram.unuse();
}
```

Framework Commands

- program.use()
  - Set which shader program the pipeline will use to draw geometry
- GLUniform.*setST(program.getUniform("VP"), …)*
  - Tell shader program to use the specified transformation as “VP”
- GLUniform.*set(program.getUniform("uGridColor"), …)*
  - Tell shader program to use the specified color as “uGridColor”
- GLProgram.unuse()
  - Tell OpenGL we are done drawing for now
- **Each of these has OpenGL commands under the hood**

Framework Commands

- vb.useAsAttrib(program.getAttribute("vPos"))
  - Tell shader program to use “vb” as vertex buffer and access vertex position using “vPos”

- ib.bind(), ib.unbind()
  - Bind (and unbind) the index buffer to tell OpenGL about how we use the vertices in the vertex buffer

Why Have a Framework?

- You write:

```
vb.useAsAttrib(program.getAttribute("vPos"));
```

- Framework does:

```
GL15.glBindBuffer(GL11.GL_ARRAY_BUFFER, vb.id);
GL15.glEnableVertexAttribArray(program.getAttribute("vPos"));
GL15.glVertexAttribPointer(program.getAttribute("vPos"), componentCount,
componentFormat, norm, elementByteSize, offset * elementByteSize);
```

- Annoying to retype full sequence of commands for every draw

Framework and GL Resources

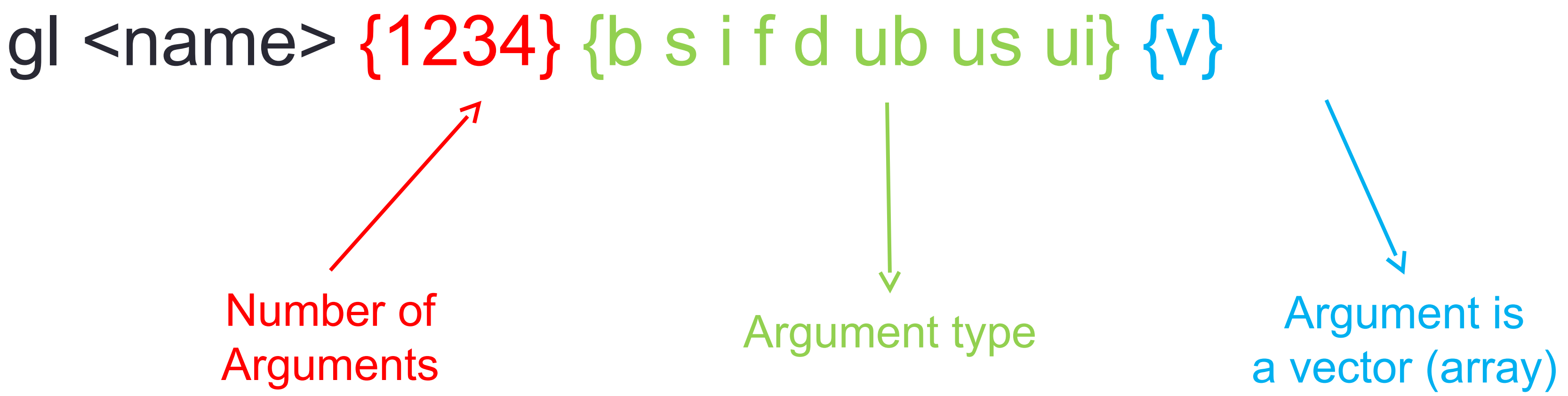
- OpenGL API has “objects” that hold rendering resources
  - Geometry, textures, shader programs, etc.
- Framework represents these with Java classes
  - GLProgram (shader programs)
  - GLBuffer (used to specify geometry)
- Constructing an object creates OpenGL resource
  - Object's data lives in GPU memory
  - Allows faster access while rendering

OpenGL Commands

- Get OpenGL context that is already initialized
- API calls: glxx.glSomeCommandName
- GL11.glClearColor(0.0f, 0.0f, 0.0f, 1.0f)
  - Set black as the color to use when clearing the screen
- GL11.glClear(GL11.GL_COLOR_BUFFER_BIT)
  - Clear the display buffer using the color given by glClearColor
- GL11.glDrawElements(…)
  - Draw primitives (now triangles)

Command Naming

- In C,
  - commands = functions
  - No two functions can have the same name
  - Some commands take different arguments but do the same thing
- All are commands of the form:



gl <name> {1234} {b s i f d ub us ui} {v}

Number of Arguments

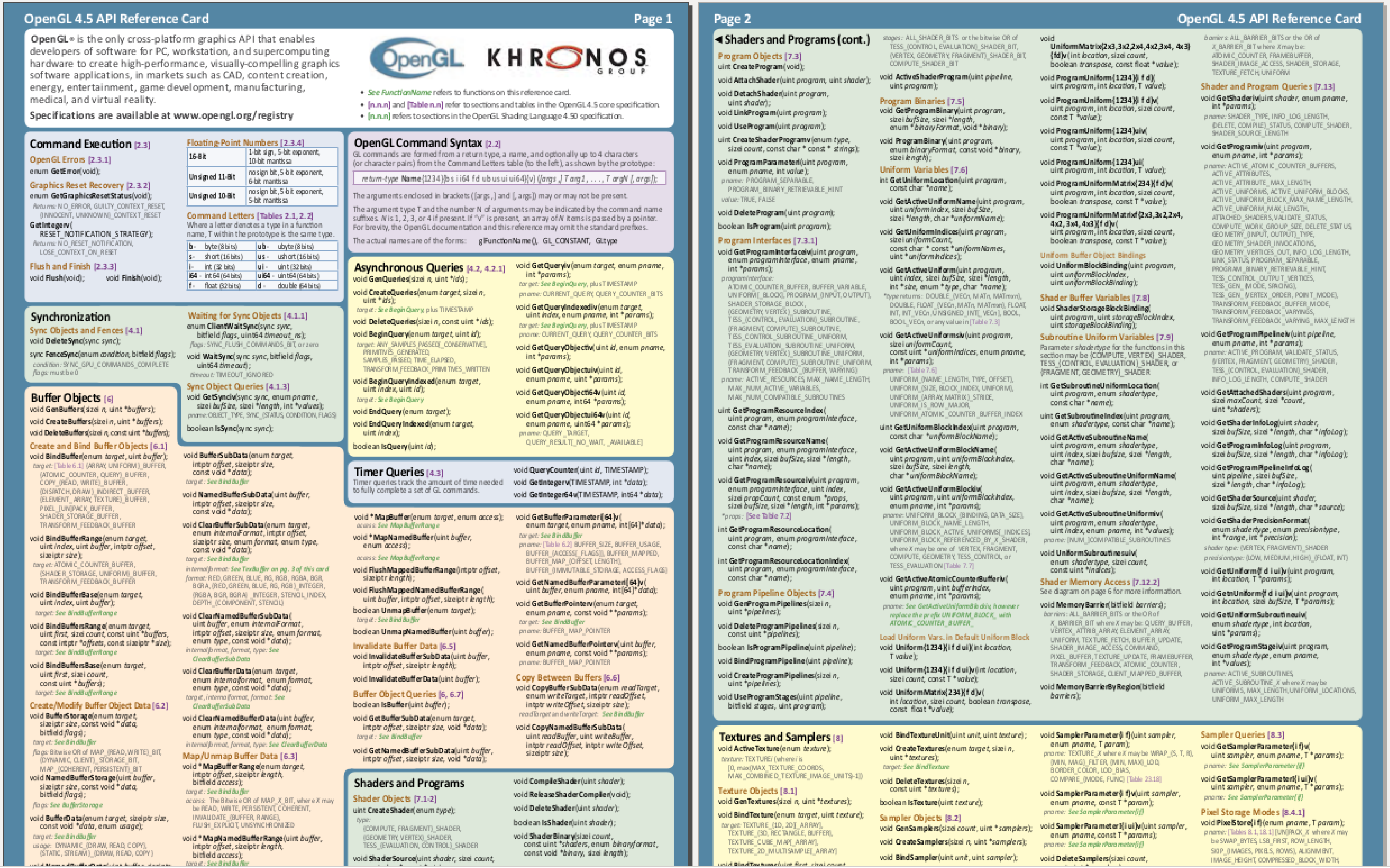
Argument type

Argument is a vector (array)

Argument Types in Command Names

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned long	GLuint, GLenum, GLbitfield

OpenGL/GLSL reference card



OpenGL 4.5 API Reference Card

Page 1

OpenGL® is the only cross-platform graphics API that enables developers of software for PC, workstation, and supercomputing hardware to create high-performance, visually-compelling graphics software applications, in markets such as CAD, content creation, energy, entertainment, game development, manufacturing, medical, and virtual reality.

Specifications are available at www.opengl.org/registry

- See FunctionName refers to functions on this reference card.
- [n.n.n] and [Table n.n] refer to sections and tables in the OpenGL 4.5 core specification.
- [n.n.n] refers to sections in the OpenGL Shading Language 4.50 specification.

Command Execution [2.3]

OpenGL Errors [2.3.1]
enum GetError(void);

Graphics Reset Recovery [2.3.2]
enum GetGraphicsResetStatus(void);
Returns: NO_ERROR, GUILTY_CONTEXT_RESET, {INNOCENT, UNKNOWN}_CONTEXT_RESET
GetIntegerv(RESET_NOTIFICATION_STRATEGY);
Returns: NO_RESET_NOTIFICATION, LOSE_CONTEXT_ON_RESET

Flush and Finish [2.3.3]
void Flush(void); void Finish(void);

Floating-Point Numbers [2.3.4]

16-Bit	1-bit sign, 5-bit exponent, 10-bit mantissa
Unsigned 11-Bit	no sign bit, 5-bit exponent, 6-bit mantissa
Unsigned 10-Bit	no sign bit, 5-bit exponent, 5-bit mantissa

Command Letters [Tables 2.1, 2.2]
Where a letter denotes a type in a function name, T within the prototype is the same type.

b -	byte (8 bits)	ub -	ubyte (8 bits)
s -	short (16 bits)	us -	ushort (16 bits)
i -	int (32 bits)	ui -	uint (32 bits)
i64 -	int64 (64 bits)	ui64 -	uint64 (64 bits)
f -	float (32 bits)	d -	double (64 bits)

OpenGL Command Syntax [2.2]

GL commands are formed from a return type, a name, and optionally up to 4 characters (or character pairs) from the Command Letters table (to the left), as shown by the prototype:

return-type Name{1234}{b s i i64 f d ub us ui ui64}{v} ([args ,] T arg1 , . . . , T argN [, args]);

The arguments enclosed in brackets ([args ,] and [, args]) may or may not be present.

The argument type T and the number N of arguments may be indicated by the command name suffixes. N is 1, 2, 3, or 4 if present. If "v" is present, an array of N items is passed by a pointer. For brevity, the OpenGL documentation and this reference may omit the standard prefixes.

The actual names are of the forms: glFunctionName(), GL_CONSTANT, GLtype

Synchronization

Sync Objects and Fences [4.1]
void DeleteSync(sync sync);
sync FenceSync(enum condition, bitfield flags);
condition: SYNC_GPU_COMMANDS_COMPLETE
flags: must be 0

Waiting for Sync Objects [4.1.1]
enum ClientWaitSync(sync sync, bitfield flags, uint64 timeout_ns);
flags: SYNC_FLUSH_COMMANDS_BIT, or zero
void WaitSync(sync sync, bitfield flags, uint64 timeout);
timeout: TIMEOUT_IGNORED

Sync Object Queries [4.1.3]
void GetSynciv(sync sync, enum pname, sizei bufSize, sizei *length, int *values);
pname: OBJECT_TYPE, SYNC_{STATUS, CONDITION, FLAGS}
boolean IsSync(sync sync);

Asynchronous Queries [4.2, 4.2.1]

void GenQueries(sizei n, uint *ids);
void CreateQueries(enum target, sizei n, uint *ids);
target: See BeginQuery, plus TIMESTAMP
void DeleteQueries(sizei n, const uint *ids);
void BeginQuery(enum target, uint id);
target: ANY_SAMPLES_PASSED[_CONSERVATIVE], PRIMITIVES_GENERATED, SAMPLES_PASSED, TIME_ELAPSED, TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN
void BeginQueryIndexed(enum target, uint index, uint id);
target: See BeginQuery
void EndQuery(enum target);
void EndQueryIndexed(enum target, uint index);
boolean IsQuery(uint id);
void GetQueryiv(enum target, enum pname, int *params);
target: See BeginQuery, plus TIMESTAMP
pname: CURRENT_QUERY, QUERY_COUNTER_BITS
void GetQueryIndexediv(enum target, uint index, enum pname, int *params);
target: See BeginQuery, plus TIMESTAMP
pname: CURRENT_QUERY, QUERY_COUNTER_BITS
void GetQueryObjectiv(uint id, enum pname, int *params);
void GetQueryObjectuiv(uint id, enum pname, uint *params);
void GetQueryObjecti64v(uint id, enum pname, int64 *params);
void GetQueryObjectui64v(uint id, enum pname, uint64 *params);
pname: QUERY_TARGET, QUERY_RESULT[_NO_WAIT, _AVAILABLE]

Timer Queries [4.3]

Timer queries track the amount of time needed to fully complete a set of GL commands.

void QueryCounter(uint id, TIMESTAMP);
void GetIntegerv(TIMESTAMP, int *data);
void GetInteger64v(TIMESTAMP, int64 *data);

Buffer Objects [6]

void GenBuffers(sizei n, uint *buffers);
void CreateBuffers(sizei n, uint *buffers);
void DeleteBuffers(sizei n, const uint *buffers);

Create and Bind Buffer Objects [6.1]
void BindBuffer(enum target, uint buffer);
target: [Table 6.1] {ARRAY, UNIFORM}_BUFFER, {ATOMIC_COUNTER, QUERY}_BUFFER, COPY_{READ, WRITE}_BUFFER, {DISPATCH, DRAW}_INDIRECT_BUFFER, {ELEMENT_ARRAY, TEXTURE}_BUFFER, PIXEL_[UN]PACK_BUFFER, SHADER_STORAGE_BUFFER, TRANSFORM_FEEDBACK_BUFFER
void BindBufferRange(enum target, uint index, uint buffer, intptr offset, sizeiptr size);
target: ATOMIC_COUNTER_BUFFER, {SHADER_STORAGE, UNIFORM}_BUFFER, TRANSFORM_FEEDBACK_BUFFER
void BindBufferBase(enum target, uint index, uint buffer);
target: See BindBufferRange
void BindBuffersRange(enum target, uint first, sizei count, const uint *buffers, const intptr *offsets, const sizeiptr *size);
target: See BindBufferRange
void BindBuffersBase(enum target, uint first, sizei count, const uint *buffers);
target: See BindBufferRange

Create/Modify Buffer Object Data [6.2]
void BufferStorage(enum target, sizeiptr size, const void *data, bitfield flags);
target: See BindBuffer
flags: Bitwise OR of MAP_{READ, WRITE}_BIT, {DYNAMIC, CLIENT}_STORAGE_BIT, MAP_{COHERENT, PERSISTENT}_BIT
void NamedBufferStorage(uint buffer, sizeiptr size, const void *data, bitfield flags);
flags: See BufferStorage
void BufferData(enum target, sizeiptr size, const void *data, enum usage);
target: See BindBuffer
usage: DYNAMIC_{DRAW, READ, COPY}, {STATIC, STREAM}_{DRAW, READ, COPY}

void BufferSubData(enum target, intptr offset, sizeiptr size, const void *data);
target: See BindBuffer
void NamedBufferSubData(uint buffer, intptr offset, sizeiptr size, const void *data);
void ClearBufferSubData(enum target, enum internalFormat, intptr offset, sizeiptr size, enum format, enum type, const void *data);
target: See BindBuffer
internalformat: See TexBuffer on pg. 3 of this card
format: RED, GREEN, BLUE, RG, RGB, RGBA, BGR, BGRA, {RED, GREEN, BLUE, RG, RGB}_INTEGER, {RGBA, BGR, BGRA}_INTEGER, STENCIL_INDEX, DEPTH_{COMPONENT, STENCIL}
void ClearNamedBufferSubData(uint buffer, enum internalFormat, intptr offset, sizeiptr size, enum format, enum type, const void *data);
internalformat, format, type: See ClearBufferSubData
void ClearBufferData(enum target, enum internalformat, enum format, enum type, const void *data);
target, internalformat, format: See ClearBufferSubData
void ClearNamedBufferData(uint buffer, enum internalformat, enum format, enum type, const void *data);
internalformat, format, type: See ClearBufferData

Map/Unmap Buffer Data [6.3]
void *MapBufferRange(enum target, intptr offset, sizeiptr length, bitfield access);
target: See BindBuffer
access: The Bitwise OR of MAP_X_BIT, where X may be READ, WRITE, PERSISTENT, COHERENT, INVALIDATE_{BUFFER, RANGE}, FLUSH_EXPLICIT, UNSYNCHRONIZED
void *MapNamedBufferRange(uint buffer, intptr offset, sizeiptr length, bitfield access);
target: See BindBuffer

void *MapBuffer(enum target, enum access);
access: See MapBufferRange
void *MapNamedBuffer(uint buffer, enum access);
access: See MapBufferRange
void FlushMappedBufferRange(intptr offset, sizeiptr length);
void FlushMappedNamedBufferRange(uint buffer, intptr offset, sizeiptr length);
boolean UnmapBuffer(enum target);
target: See BindBuffer
boolean UnmapNamedBuffer(uint buffer);

Invalidate Buffer Data [6.5]
void InvalidateBufferSubData(uint buffer, intptr offset, sizeiptr length);
void InvalidateBufferData(uint buffer);

Buffer Object Queries [6, 6.7]
boolean IsBuffer(uint buffer);
void GetBufferSubData(enum target, intptr offset, sizeiptr size, void *data);
target: See BindBuffer
void GetNamedBufferSubData(uint buffer, intptr offset, sizeiptr size, void *data);

void GetBufferParameteri[64]v(enum target, enum pname, int[64]*data);
target: See BindBuffer
pname: [Table 6.2] BUFFER_SIZE, BUFFER_USAGE, BUFFER_{ACCESS[_FLAGS]}, BUFFER_MAPPED, BUFFER_MAP_{OFFSET, LENGTH}, BUFFER_{IMMUTABLE_STORAGE, ACCESS_FLAGS}
void GetNamedBufferParameteri[64]v(uint buffer, enum pname, int[64]*data);
void GetBufferPointerv(enum target, enum pname, const void **params);
target: See BindBuffer
pname: BUFFER_MAP_POINTER
void GetNamedBufferPointerv(uint buffer, enum pname, const void **params);
pname: BUFFER_MAP_POINTER

Copy Between Buffers [6.6]
void CopyBufferSubData(enum readTarget, enum writeTarget, intptr readOffset, intptr writeOffset, sizeiptr size);
readTarget and writeTarget: See BindBuffer
void CopyNamedBufferSubData(uint readBuffer, uint writeBuffer, intptr readOffset, intptr writeOffset, sizeiptr size);

Shaders and Programs

Shader Objects [7.1-2]
uint CreateShader(enum type);
type: {COMPUTE, FRAGMENT}_SHADER, {GEOMETRY, VERTEX}_SHADER, TESS_{EVALUATION, CONTROL}_SHADER
void ShaderSource(uint shader, sizei count,

void CompileShader(uint shader);
void ReleaseShaderCompiler(void);
void DeleteShader(uint shader);
boolean IsShader(uint shader);
void ShaderBinary(sizei count, const uint *shaders, enum binaryformat, const void *binary, sizei length);

Page 2

OpenGL 4.5 API Reference Card

◄ Shaders and Programs (cont.)

Program Objects [7.3]
uint CreateProgram(void);
void AttachShader(uint program, uint shader);
void DetachShader(uint program, uint shader);
void LinkProgram(uint program);
void UseProgram(uint program);
uint CreateShaderProgramv(enum type, sizei count, const char * const * strings);
void ProgramParameteri(uint program, enum pname, int value);
pname: PROGRAM_SEPARABLE, PROGRAM_BINARY_RETRIEVABLE_HINT
value: TRUE, FALSE
void DeleteProgram(uint program);
boolean IsProgram(uint program);

Program Interfaces [7.3.1]
void GetProgramInterfaceiv(uint program, enum programInterface, enum pname, int *params);
programInterface: ATOMIC_COUNTER_BUFFER, BUFFER_VARIABLE, UNIFORM[_BLOCK], PROGRAM_{INPUT, OUTPUT}, SHADER_STORAGE_BLOCK, {GEOMETRY, VERTEX}_SUBROUTINE, TESS_{CONTROL, EVALUATION}_SUBROUTINE, {FRAGMENT, COMPUTE}_SUBROUTINE, TESS_CONTROL_SUBROUTINE_UNIFORM, TESS_EVALUATION_SUBROUTINE_UNIFORM, {GEOMETRY, VERTEX}_SUBROUTINE_UNIFORM, {FRAGMENT, COMPUTE}_SUBROUTINE_UNIFORM, TRANSFORM_FEEDBACK_{BUFFER, VARYING}
pname: ACTIVE_RESOURCES, MAX_NAME_LENGTH, MAX_NUM_ACTIVE_VARIABLES, MAX_NUM_COMPATIBLE_SUBROUTINES
uint GetProgramResourceIndex(uint program, enum programInterface, const char *name);
void GetProgramResourceName(uint program, enum programInterface, uint index, sizei bufSize, sizei *length, char *name);
void GetProgramResourceiv(uint program, enum programInterface, uint index, sizei propCount, const enum *props, sizei bufSize, sizei *length, int *params);
*props: [See Table 7.2]
int GetProgramResourceLocation(uint program, enum programInterface, const char *name);
int GetProgramResourceLocationIndex(uint program, enum programInterface, const char *name);

Program Pipeline Objects [7.4]
void GenProgramPipelines(sizei n, uint *pipelines);
void DeleteProgramPipelines(sizei n, const uint *pipelines);
boolean IsProgramPipeline(uint pipeline);
void BindProgramPipeline(uint pipeline);
void CreateProgramPipelines(sizei n, uint *pipelines);
void UseProgramStages(uint pipeline, bitfield stages, uint program);

stages: ALL_SHADER_BITS or the bitwise OR of TESS_{CONTROL, EVALUATION}_SHADER_BIT, {VERTEX, GEOMETRY, FRAGMENT}_SHADER_BIT, COMPUTE_SHADER_BIT
void ActiveShaderProgram(uint pipeline, uint program);

Program Binaries [7.5]
void GetProgramBinary(uint program, sizei bufSize, sizei *length, enum *binaryFormat, void *binary);
void ProgramBinary(uint program, enum binaryFormat, const void *binary, sizei length);

Uniform Variables [7.6]
int GetUniformLocation(uint program, const char *name);
void GetActiveUniformName(uint program, uint uniformIndex, sizei bufSize, sizei *length, char *uniformName);
void GetUniformIndices(uint program, sizei uniformCount, const char * const *uniformNames, uint *uniformIndices);
void GetActiveUniform(uint program, uint index, sizei bufSize, sizei *length, int *size, enum *type, char *name);
*type returns: DOUBLE_{VECn, MATn, MATnxm}, DOUBLE, FLOAT_{VECn, MATn, MATnxm}, FLOAT, INT, INT_VECn, UNSIGNED_INT[_VECn], BOOL, BOOL_VECn, or any value in [Table 7.3]
void GetActiveUniformsiv(uint program, sizei uniformCount, const uint *uniformIndices, enum pname, int *params);
pname: [Table 7.6] UNIFORM_{NAME_LENGTH, TYPE, OFFSET}, UNIFORM_{SIZE, BLOCK_INDEX, UNIFORM}, UNIFORM_{ARRAY, MATRIX}_STRIDE, UNIFORM_IS_ROW_MAJOR, UNIFORM_ATOMIC_COUNTER_BUFFER_INDEX
uint GetUniformBlockIndex(uint program, const char *uniformBlockName);
void GetActiveUniformBlockName(uint program, uint uniformBlockIndex, sizei bufSize, sizei length, char *uniformBlockName);
void GetActiveUniformBlockiv(uint program, uint uniformBlockIndex, enum pname, int *params);
pname: UNIFORM_BLOCK_{BINDING, DATA_SIZE}, UNIFORM_BLOCK_NAME_LENGTH, UNIFORM_BLOCK_ACTIVE_UNIFORMS[_INDICES], UNIFORM_BLOCK_REFERENCED_BY_X_SHADER, where X may be one of VERTEX, FRAGMENT, COMPUTE, GEOMETRY, TESS_CONTROL, or TESS_EVALUATION [Table 7.7]
void GetActiveAtomicCounterBufferiv(uint program, uint bufferIndex, enum pname, int *params);
pname: See GetActiveUniformBlockiv, however replace the prefix UNIFORM_BLOCK_ with ATOMIC_COUNTER_BUFFER_

Load Uniform Vars. In Default Uniform Block
void Uniform{1234}{i f d ui}(int location, T value);
void Uniform{1234}{i f d ui}v(int location, sizei count, const T *value);
void UniformMatrix{234}{f d}v(int location, sizei count, boolean transpose, const float *value);

void UniformMatrix{2x3,3x2,2x4,4x2,3x4, 4x3}{fd}v(int location, sizei count, boolean transpose, const float *value);
void ProgramUniform{1234}{i f d}(uint program, int location, T value);
void ProgramUniform{1234}{i f d}v(uint program, int location, sizei count, const T *value);
void ProgramUniform{1234}uiv(uint program, int location, sizei count, const T *value);
void ProgramUniform{1234}ui(uint program, int location, T value);
void ProgramUniformMatrix{234}{f d}v(uint program, int location, sizei count, boolean transpose, const T *value);
void ProgramUniformMatrixf{2x3,3x2,2x4, 4x2, 3x4, 4x3}{f d}v(uint program, int location, sizei count, boolean transpose, const T *value);

Uniform Buffer Object Bindings
void UniformBlockBinding(uint program, uint uniformBlockIndex, uint uniformBlockBinding);

Shader Buffer Variables [7.8]
void ShaderStorageBlockBinding(uint program, uint storageBlockIndex, uint storageBlockBinding);

Subroutine Uniform Variables [7.9]
Parameter shadertype for the functions in this section may be {COMPUTE, VERTEX}_SHADER, TESS_{CONTROL, EVALUATION}_SHADER, or {FRAGMENT, GEOMETRY}_SHADER
int GetSubroutineUniformLocation(uint program, enum shadertype, const char *name);
uint GetSubroutineIndex(uint program, enum shadertype, const char *name);
void GetActiveSubroutineName(uint program, enum shadertype, uint index, sizei bufsize, sizei *length, char *name);
void GetActiveSubroutineUniformName(uint program, enum shadertype, uint index, sizei bufsize, sizei *length, char *name);
void GetActiveSubroutineUniformiv(uint program, enum shadertype, uint index, enum pname, int *values);
pname: [NUM_]COMPATIBLE_SUBROUTINES
void UniformSubroutinesuiv(enum shadertype, sizei count, const uint *indices);

Shader Memory Access [7.12.2]
See diagram on page 6 for more information.
void MemoryBarrier(bitfield barriers);
barriers: ALL_BARRIER_BITS or the OR of X_BARRIER_BIT where X may be: QUERY_BUFFER, VERTEX_ATTRIB_ARRAY, ELEMENT_ARRAY, UNIFORM, TEXTURE_FETCH, BUFFER_UPDATE, SHADER_IMAGE_ACCESS, COMMAND, PIXEL_BUFFER, TEXTURE_UPDATE, FRAMEBUFFER, TRANSFORM_FEEDBACK, ATOMIC_COUNTER, SHADER_STORAGE, CLIENT_MAPPED_BUFFER,
void MemoryBarrierByRegion(bitfield barriers);

barriers: ALL_BARRIER_BITS or the OR of X_BARRIER_BIT where X may be: ATOMIC_COUNTER, FRAMEBUFFER, SHADER_IMAGE_ACCESS, SHADER_STORAGE, TEXTURE_FETCH, UNIFORM

Shader and Program Queries [7.13]
void GetShaderiv(uint shader, enum pname, int *params);
pname: SHADER_TYPE, INFO_LOG_LENGTH, {DELETE, COMPILE}_STATUS, COMPUTE_SHADER, SHADER_SOURCE_LENGTH
void GetProgramiv(uint program, enum pname, int *params);
pname: ACTIVE_ATOMIC_COUNTER_BUFFERS, ACTIVE_ATTRIBUTES, ACTIVE_ATTRIBUTE_MAX_LENGTH, ACTIVE_UNIFORMS, ACTIVE_UNIFORM_BLOCKS, ACTIVE_UNIFORM_BLOCK_MAX_NAME_LENGTH, ACTIVE_UNIFORM_MAX_LENGTH, ATTACHED_SHADERS, VALIDATE_STATUS, COMPUTE_WORK_GROUP_SIZE, DELETE_STATUS, GEOMETRY_{INPUT, OUTPUT}_TYPE, GEOMETRY_SHADER_INVOCATIONS, GEOMETRY_VERTICES_OUT, INFO_LOG_LENGTH, LINK_STATUS, PROGRAM_SEPARABLE, PROGRAM_BINARY_RETRIEVABLE_HINT, TESS_CONTROL_OUTPUT_VERTICES, TESS_GEN_{MODE, SPACING}, TESS_GEN_{VERTEX_ORDER, POINT_MODE}, TRANSFORM_FEEDBACK_BUFFER_MODE, TRANSFORM_FEEDBACK_VARYINGS, TRANSFORM_FEEDBACK_VARYING_MAX_LENGTH
void GetProgramPipelineiv(uint pipeline, enum pname, int *params);
pname: ACTIVE_PROGRAM, VALIDATE_STATUS, {VERTEX, FRAGMENT, GEOMETRY}_SHADER, TESS_{CONTROL, EVALUATION}_SHADER, INFO_LOG_LENGTH, COMPUTE_SHADER
void GetAttachedShaders(uint program, sizei maxCount, sizei *count, uint *shaders);
void GetShaderInfoLog(uint shader, sizei bufSize, sizei *length, char *infoLog);
void GetProgramInfoLog(uint program, sizei bufSize, sizei *length, char *infoLog);
void GetProgramPipelineInfoLog(uint pipeline, sizei bufSize, sizei *length, char *infoLog);
void GetShaderSource(uint shader, sizei bufSize, sizei *length, char *source);
void GetShaderPrecisionFormat(enum shadertype, enum precisiontype, int *range, int *precision);
shadertype: {VERTEX, FRAGMENT}_SHADER
precisiontype: {LOW, MEDIUM, HIGH}_{FLOAT, INT}
void GetUniform{f d i ui}v(uint program, int location, T *params);
void GetnUniform{f d i ui}v(uint program, int location, sizei bufSize, T *params);
void GetUniformSubroutineuiv(enum shadertype, int location, uint *params);
void GetProgramStageiv(uint program, enum shadertype, enum pname, int *values);
pname: ACTIVE_SUBROUTINES, ACTIVE_SUBROUTINE_X where X may be UNIFORMS, MAX_LENGTH, UNIFORM_LOCATIONS, UNIFORM_MAX_LENGTH

Textures and Samplers [8]

void ActiveTexture(enum texture);
texture: TEXTUREi (where i is [0, max(MAX_TEXTURE_COORDS, MAX_COMBINED_TEXTURE_IMAGE_UNITS)-1])

Texture Objects [8.1]
void GenTextures(sizei n, uint *textures);
void BindTexture(enum target, uint texture);
target: TEXTURE_{1D, 2D}[_ARRAY], TEXTURE_{3D, RECTANGLE, BUFFER}, TEXTURE_CUBE_MAP[_ARRAY], TEXTURE_2D_MULTISAMPLE[_ARRAY]

void BindTextureUnit(uint unit, uint texture);
void CreateTextures(enum target, sizei n, uint *textures);
target: See BindTexture
void DeleteTextures(sizei n, const uint *textures);
boolean IsTexture(uint texture);

Sampler Objects [8.2]
void GenSamplers(sizei count, uint *samplers);
void CreateSamplers(sizei n, uint *samplers);
void BindSampler(uint unit, uint sampler);

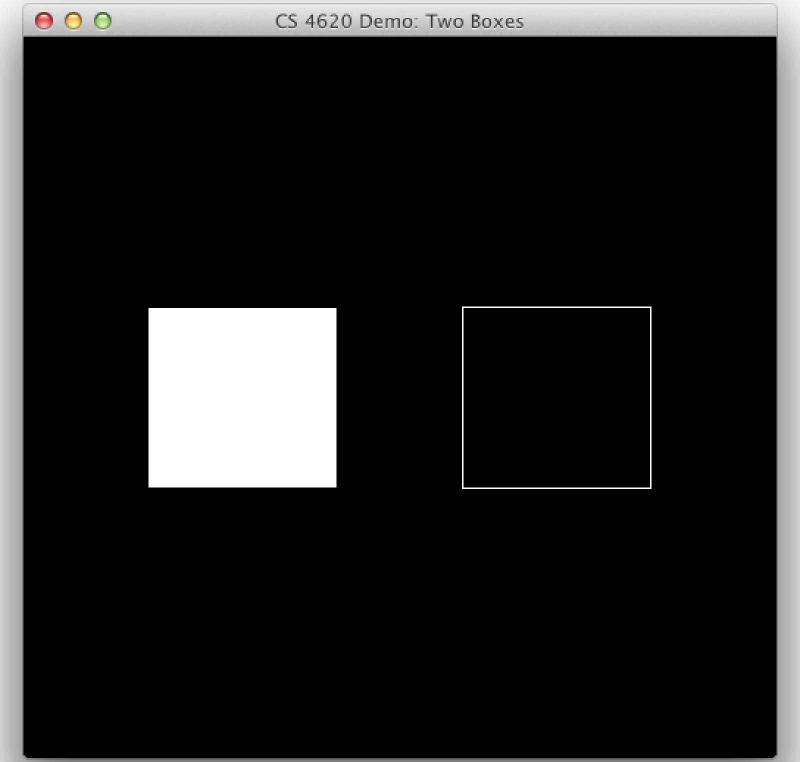
void SamplerParameter{i f}(uint sampler, enum pname, T param);
pname: TEXTURE_X where X may be WRAP_{S, T, R}, {MIN, MAG}_FILTER, {MIN, MAX}_LOD, BORDER_COLOR, LOD_BIAS, COMPARE_{MODE, FUNC} [Table 23.18]
void SamplerParameter{i f}v(uint sampler, enum pname, const T *param);
pname: See SamplerParameter{i f}
void SamplerParameterI{i ui}v(uint sampler, enum pname, const T *params);
pname: See SamplerParameter{i f}
void DeleteSamplers(sizei count,

Sampler Queries [8.3]
void GetSamplerParameter{i f}v(uint sampler, enum pname, T *params);
pname: See SamplerParameter{i f}
void GetSamplerParameterI{i ui}v(uint sampler, enum pname, T *params);
pname: See SamplerParameter{i f}

Pixel Storage Modes [8.4.1]
void PixelStore{i f}(enum pname, T param);
pname: [Tables 8.1, 18.1] [UN]PACK_X where X may be SWAP_BYTES, LSB_FIRST, ROW_LENGTH, SKIP_{IMAGES, PIXELS, ROWS}, ALIGNMENT, IMAGE_HEIGHT, COMPRESSED_BLOCK_WIDTH,

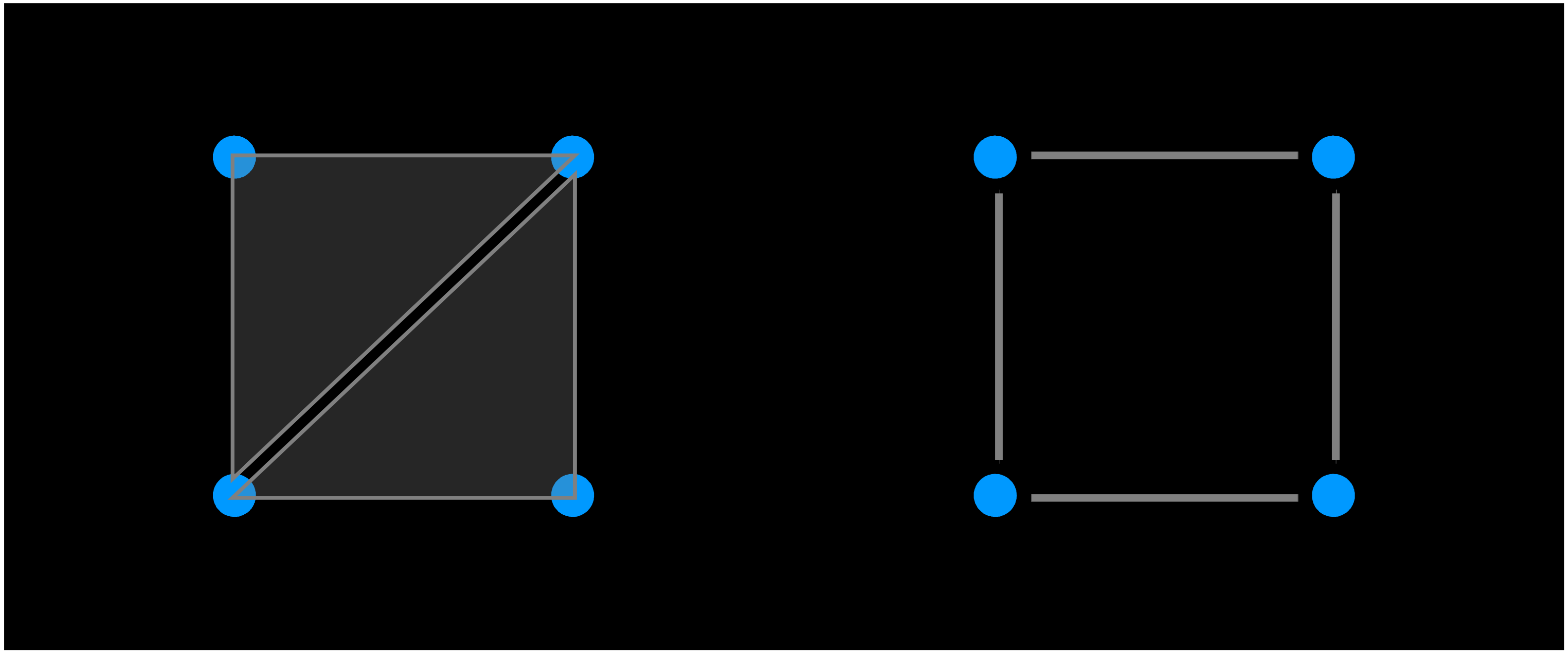
OpenGL Resources: Geometry

Demo: Two Boxes



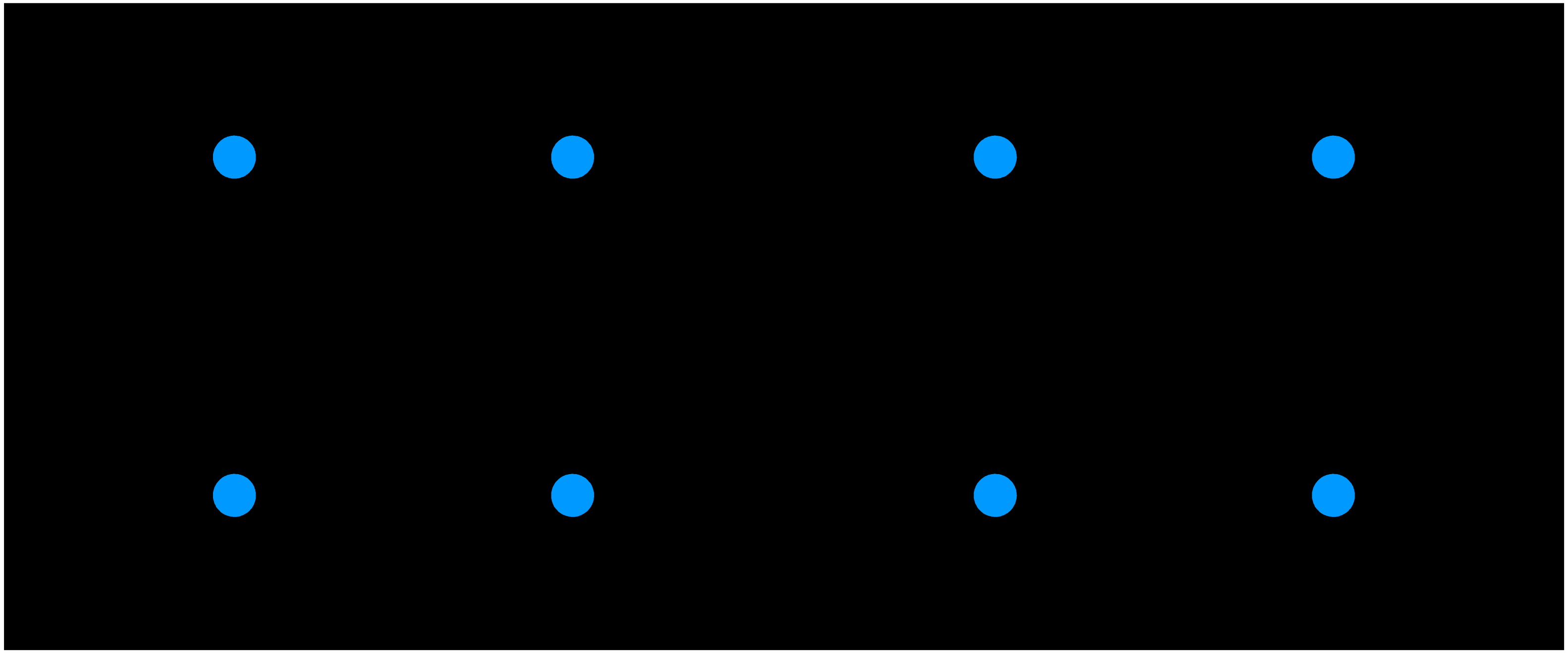
What We're Seeing

- Box on left: two triangles
- Box on right: four lines



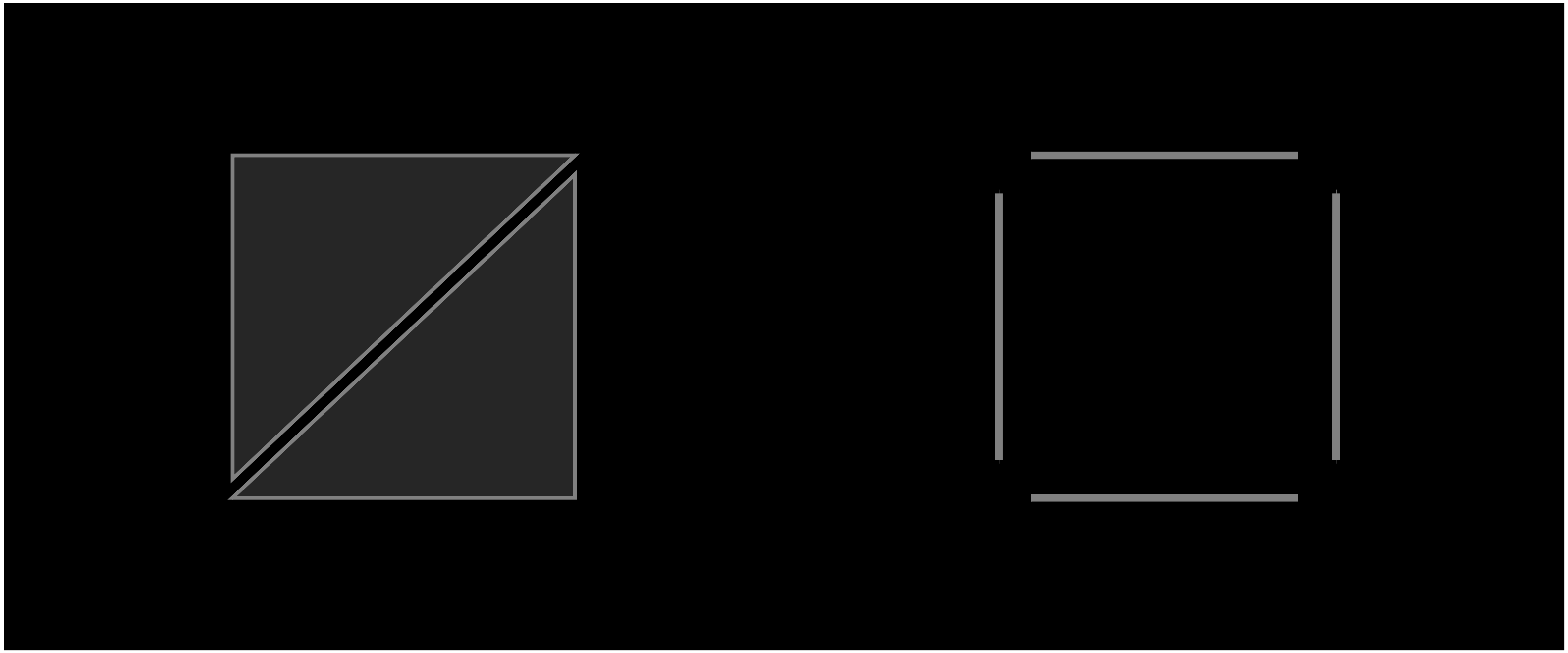
Vertices

- Foundation for all geometry
- OpenGL: specify with GLBuffer



Primitives

- Basic shapes built from vertices; e.g. triangles, lines
  - Assemble to build more complicated shapes
- OpenGL: specify both with GLBuffer



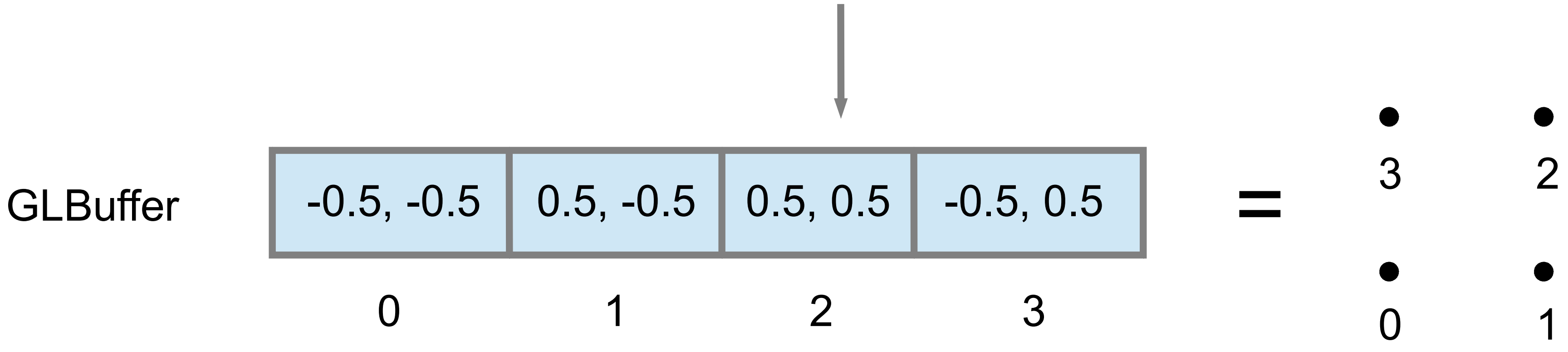
GLBuffer

- OpenGL object to store arrays like vertex positions, vertex colors, indices
- We have to specify:
  - How many component per element
    - Color: 3D vector
    - Position: 2D/3D vector
    - Index: 1D
  - Type of stored element components (int, float, double, …)
  - Array of element components
    - The stored data itself

Specifying Vertices

- GLBuffer: store sequence of vertex positions
- Info needed:
  - Array of floats representing vertices
  - How many dimensions per vertex (2D? 3D?)

([-0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, 0.5], 2 floats per vertex)



GLBuffer

-0.5, -0.5	0.5, -0.5	0.5, 0.5	-0.5, 0.5
0	1	2	3

Specifying Vertices: TwoBoxes' init()

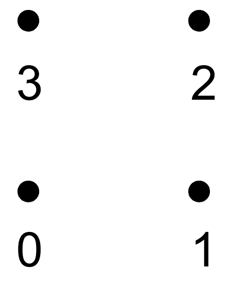
```
@Override
public void onEntry(GameTime gameTime) {
    // define vertex positions
    float [] vertexPositions = {
            -0.5f, -0.5f,      // vertex 0
             0.5f, -0.5f,      // vertex 1
             0.5f,  0.5f,      // vertex 2
            -0.5f,  0.5f       // vertex 3
    };

    GLBuffer vertexBuffer = GLBuffer.createAsVertex(
    vertexPositions, 2, BufferUsageHint.StaticDraw);
    ...
```

Grouping Vertices into Primitives

- GLBuffer gives vertices in some order

GLBuffer	-0.5, -0.5	0.5, -0.5	0.5, 0.5	-0.5, 0.5
	0	1	2	3

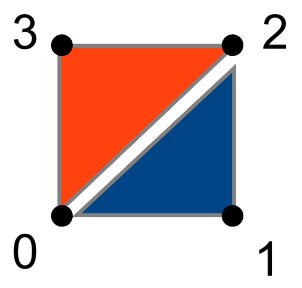
=



- Can re-order vertices to form primitives, so that:

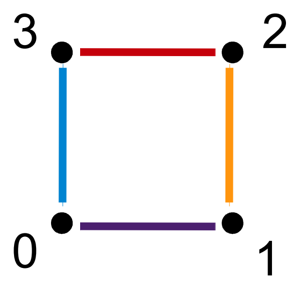
– Every three vertices form a triangle

[0, 1, 2,   0, 2, 3]



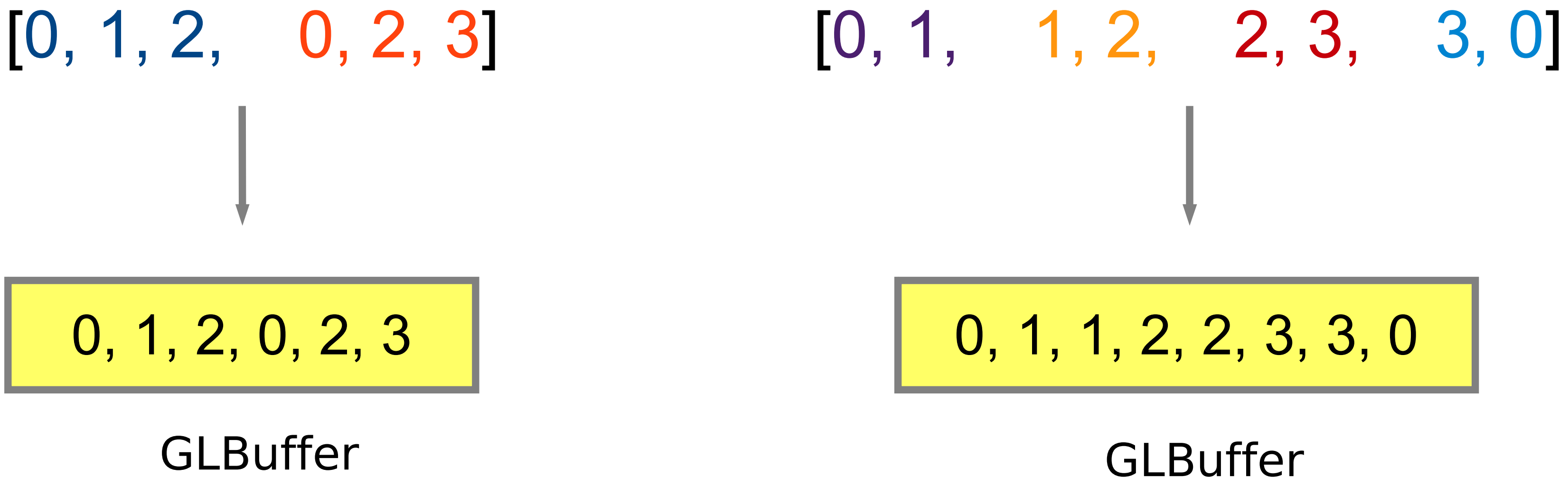
– Every two vertices form a line

[0, 1,   1, 2,   2, 3,   3, 0]



Grouping Vertices

- GLBuffer: store sequence of vertex indices
- Info needed:
  - List of integer indices



[0, 1, 2, 0, 2, 3]

0, 1, 2, 0, 2, 3

GLBuffer

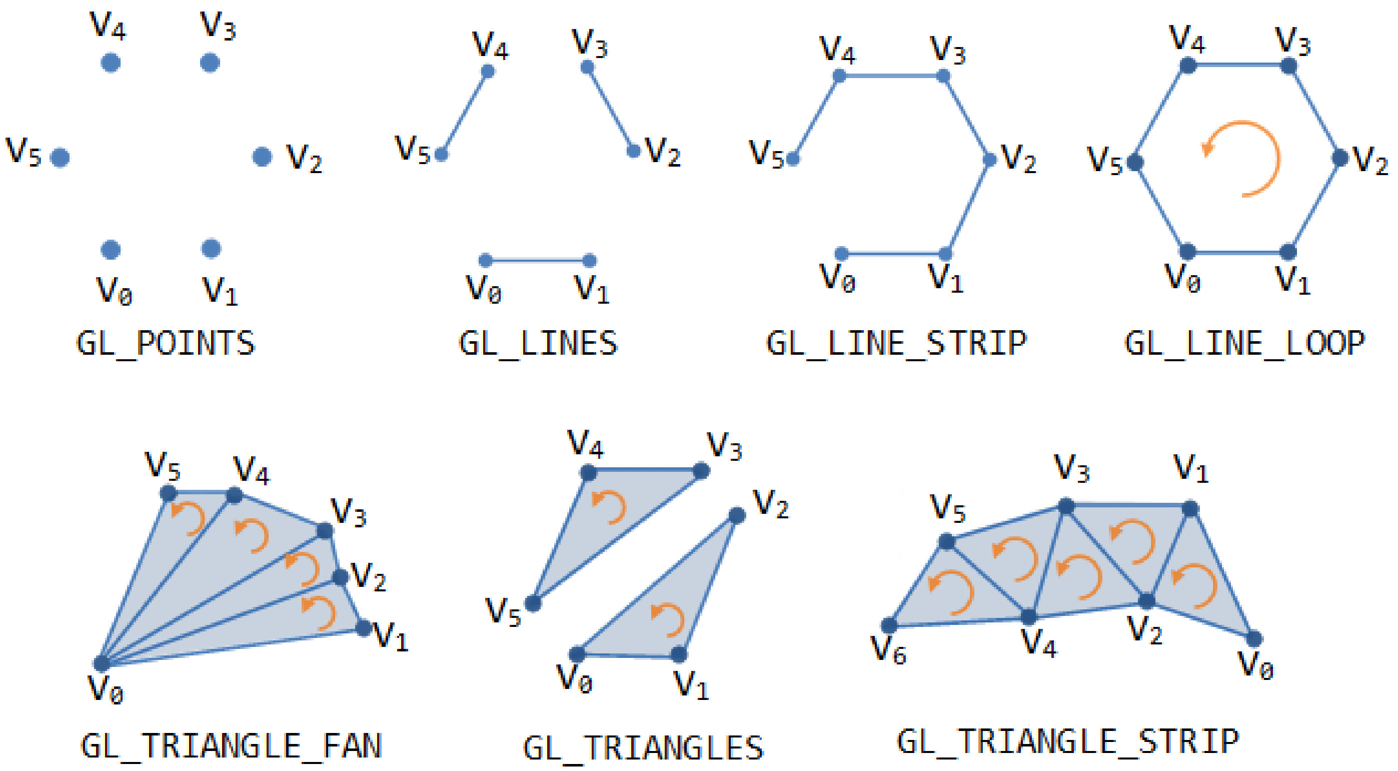
[0, 1, 1, 2, 2, 3, 3, 0]

0, 1, 1, 2, 2, 3, 3, 0

GLBuffer

Ways to Group: GL Primitives

- OpenGL declares several primitive types
- Determine how to group a sequence of vertices into primitives

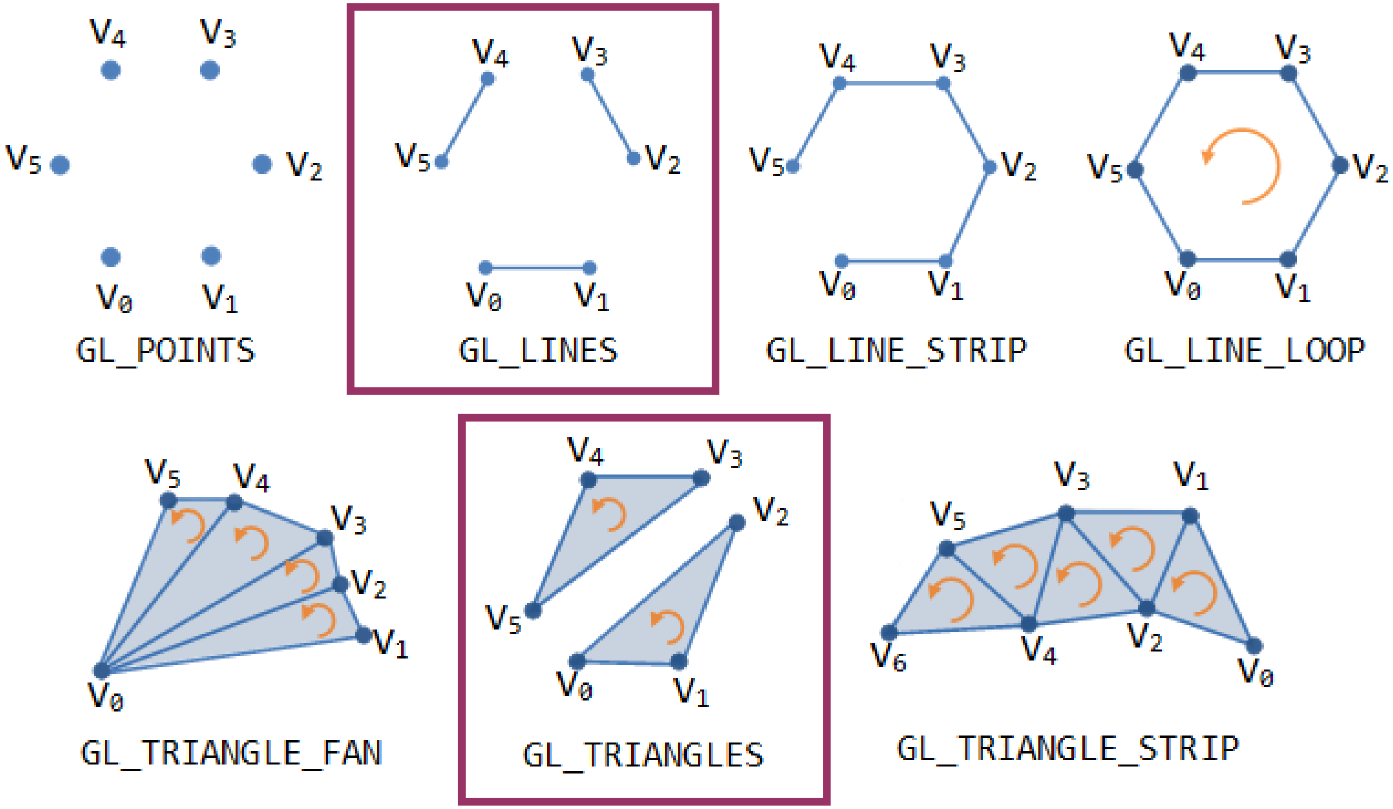

V_4 V_3 V_5 V_2 V_0 V_1
GL_POINTS
V_4 V_3 V_5 V_2 V_0 V_1
GL_LINES
V_4 V_3 V_5 V_2 V_0 V_1
GL_LINE_STRIP
V_4 V_3 V_5 V_2 V_0 V_1
GL_LINE_LOOP
V_5 V_4 V_3 V_2 V_1 V_0
GL_TRIANGLE_FAN
V_4 V_3 V_2 V_5 V_0 V_1
GL_TRIANGLES
V_3 V_1 V_5 V_6 V_4 V_2 V_0
GL_TRIANGLE_STRIP

OpenGL Primitives

(adapted from http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html)

Ways to Group: GL Primitives

- OpenGL declares several primitive types
- Determine how to group a sequence of vertices into primitives



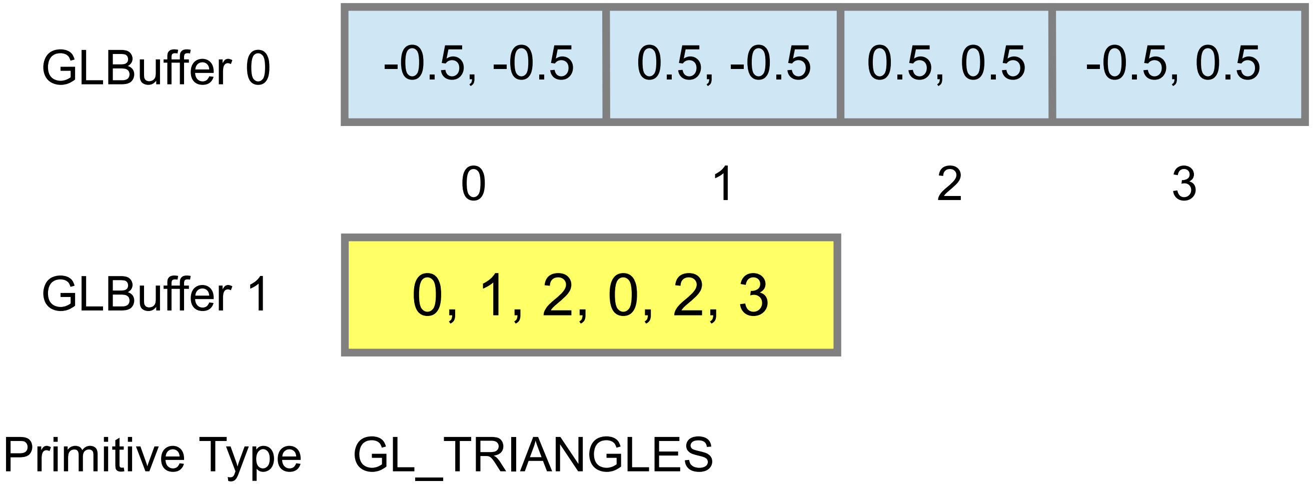
OpenGL Primitives

(adapted from http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html)

Putting it Together

- GLBuffer 0: what the vertices are
- GLBuffer 1 (index array): in what order to put them
- Primitive Type: how to group ordered vertices into primitives

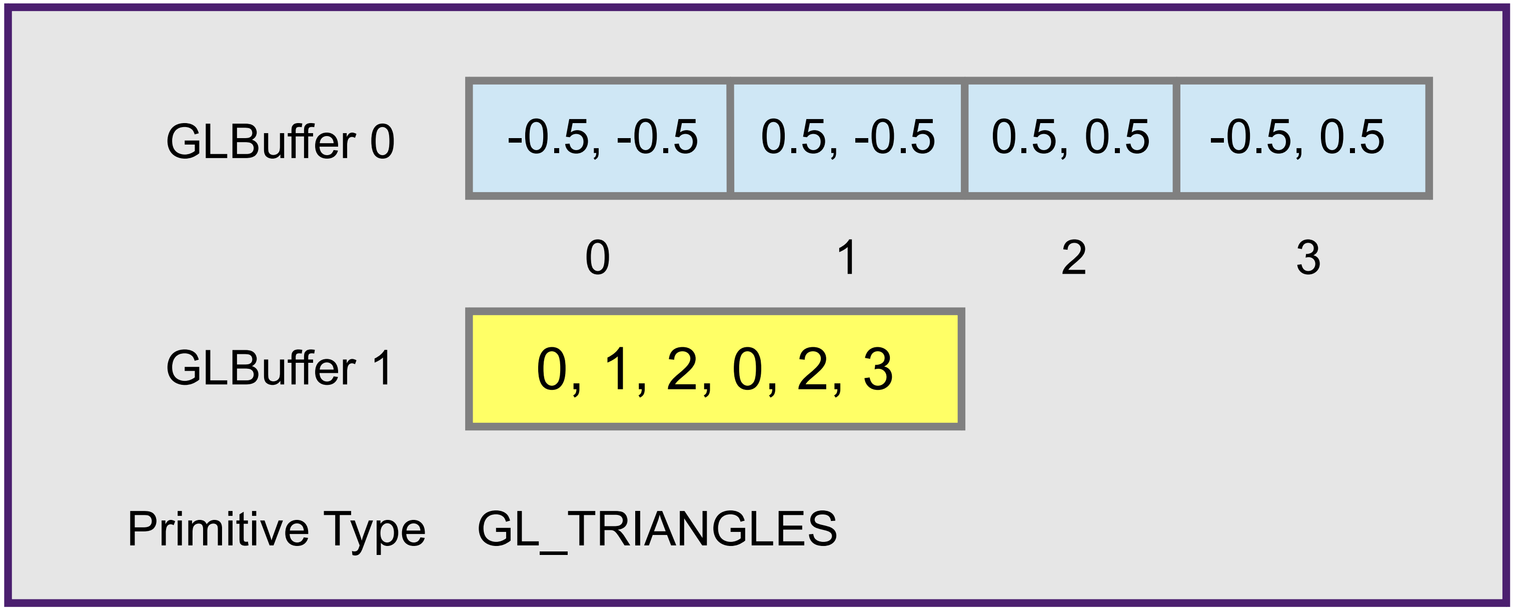
- Together, fully describes geometry



GLBuffer 0	-0.5, -0.5	0.5, -0.5	0.5, 0.5	-0.5, 0.5
	0	1	2	3
GLBuffer 1	0, 1, 2, 0, 2, 3			
Primitive Type	GL_TRIANGLES			

Putting it Together: Bindings

- Bind buffer elements to vertex attributes
- Bind indices to buffer in OpenGL
- Draw using OpenGL
- Info needed:
  - GLBuffer with vertice attributes, GLBuffer with indices, primitive type



GLBuffer 0

-0.5, -0.5	0.5, -0.5	0.5, 0.5	-0.5, 0.5
0	1	2	3

GLBuffer 1 0, 1, 2, 0, 2, 3

Primitive Type GL_TRIANGLES

Index arrays in TwoBoxes' onEntry()

```
... // earlier, filled GLBuffer vertexPositions

int [] linesIndices = {
   0, 1,
   1, 2,
   2, 3,
   3, 0
};
int [] trianglesIndices = {
   0, 1, 2,
   0, 2, 3
};
// make index buffers
ibLines = GLBuffer.createAsIndex(linesIndices,
BufferUsageHint.StaticDraw);
indexCountLines = linesIndices.length;
ibTriangles = GLBuffer.createAsIndex(trianglesIndices,
BufferUsageHint.StaticDraw);
indexCountTriangles = trianglesIndices.length;
...
```

TwoBoxes' draw()

```
...
// Use box vertices we defined before
vb.useAsAttrib(program.getAttribute("vPos"));

// Setup transformations
...
// Binding indices and drawing
                    ibTriangles.bind();
                    GL11.glDrawElements(GL11.GL_TRIANGLES,
                    indexCountTriangles,
                    GLType.UnsignedInt, 0);
                    ibTriangles.unbind();
// Setup transformations
...
// Binding indices and drawing
                    ibLines.bind();
                    GL11.glDrawElements(GL11.GL_LINES,
                    indexCountLines,
                    GLType.UnsignedInt, 0);
                    ibLines.unbind();
```

TwoBoxes' draw()

```
...
// Use box vertices we defined before
vb.useAsAttrib(program.getAttribute("vPos"));

// Setup transformations
...
// Binding indices and drawing
                    ibTriangles.bind();
                    GL11.glDrawElements(GL11.GL_TRIANGLES,
                    indexCountTriangles,
                    GLType.UnsignedInt, 0);
                    ibTriangles.unbind();
// Setup transformations
...
// Binding indices and drawing
                    ibLines.bind();
                    GL11.glDrawElements(GL11.GL_LINES,
                    indexCountLines,
                    GLType.UnsignedInt, 0);
                    ibLines.unbind();
```

- Bind index array and say what primitives we will build
- One will make triangles, the other lines

TwoBoxes' draw()

```
...
// Use box vertices we defined before
vb.useAsAttrib(program.getAttribute("vPos"));

// Setup transformations
...
// Binding indices and drawing
                    ibTriangles.bind();
                    GL11.glDrawElements(GL11.GL_TRIANGLES,
                    indexCountTriangles,
                    GLType.UnsignedInt, 0);
                    ibTriangles.unbind();
// Setup transformations
...
// Binding indices and drawing
                    ibLines.bind();
                    GL11.glDrawElements(GL11.GL_LINES,
                    indexCountLines,
                    GLType.UnsignedInt, 0);
                    ibLines.unbind();
```

- Tell OpenGL how to order the vertices when building primitives
- Triangles and lines will need different vertex orders

TwoBoxes' draw()

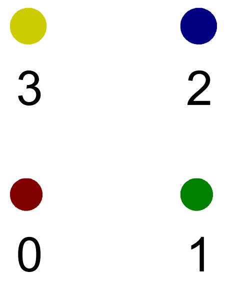
```
...
// Use box vertices we defined before
vb.useAsAttrib(program.getAttribute("vPos"));

// Setup transformations
...
// Binding indices and drawing
                    ibTriangles.bind();
                    GL11.glDrawElements(GL11.GL_TRIANGLES,
                    indexCountTriangles,
                    GLType.UnsignedInt, 0);
                    ibTriangles.unbind();
// Setup transformations
...
// Binding indices and drawing
                    ibLines.bind();
                    GL11.glDrawElements(GL11.GL_LINES,
                    indexCountLines,
                    GLType.UnsignedInt, 0);
                    ibLines.unbind();
```

- Both boxes use the same buffer with four vertices

Vertices are More than Positions

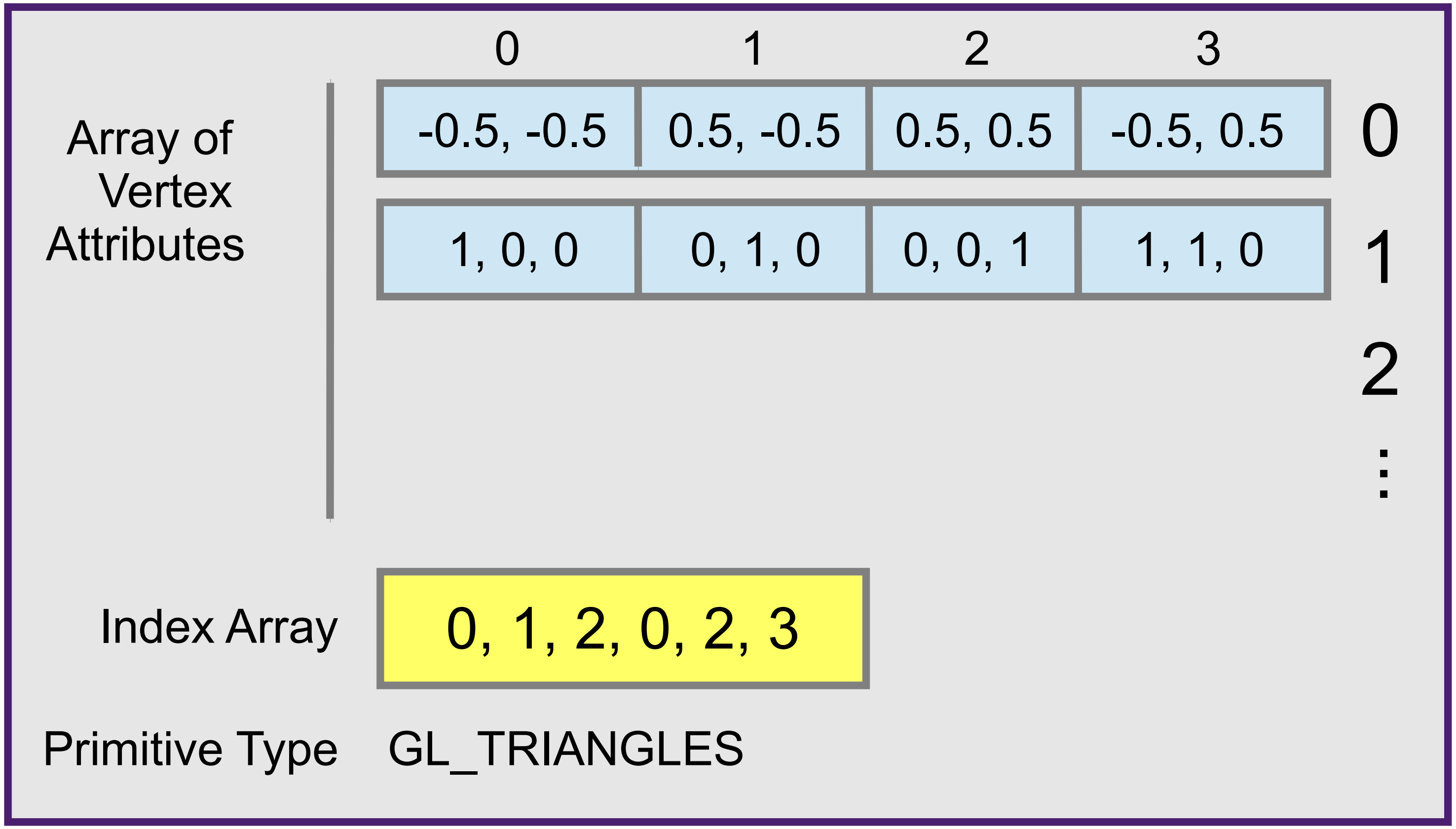
- A vertex has a position
- But it can also have other attributes:
  - Normal vector
  - Color
  - Texture coordinate
  - etc.
- Use multiple GLBuffers to store this info

	0	1	2	3
Positions	-0.5, -0.5	0.5, -0.5	0.5, 0.5	-0.5, 0.5
Colors	1, 0, 0	0, 1, 0	0, 0, 1	1, 1, 0



Multiple Vertex Attributes

- We can bind a GLBuffer to a vertex attribute
- We can use multiple attributes per vertex



Array of Vertex Attributes

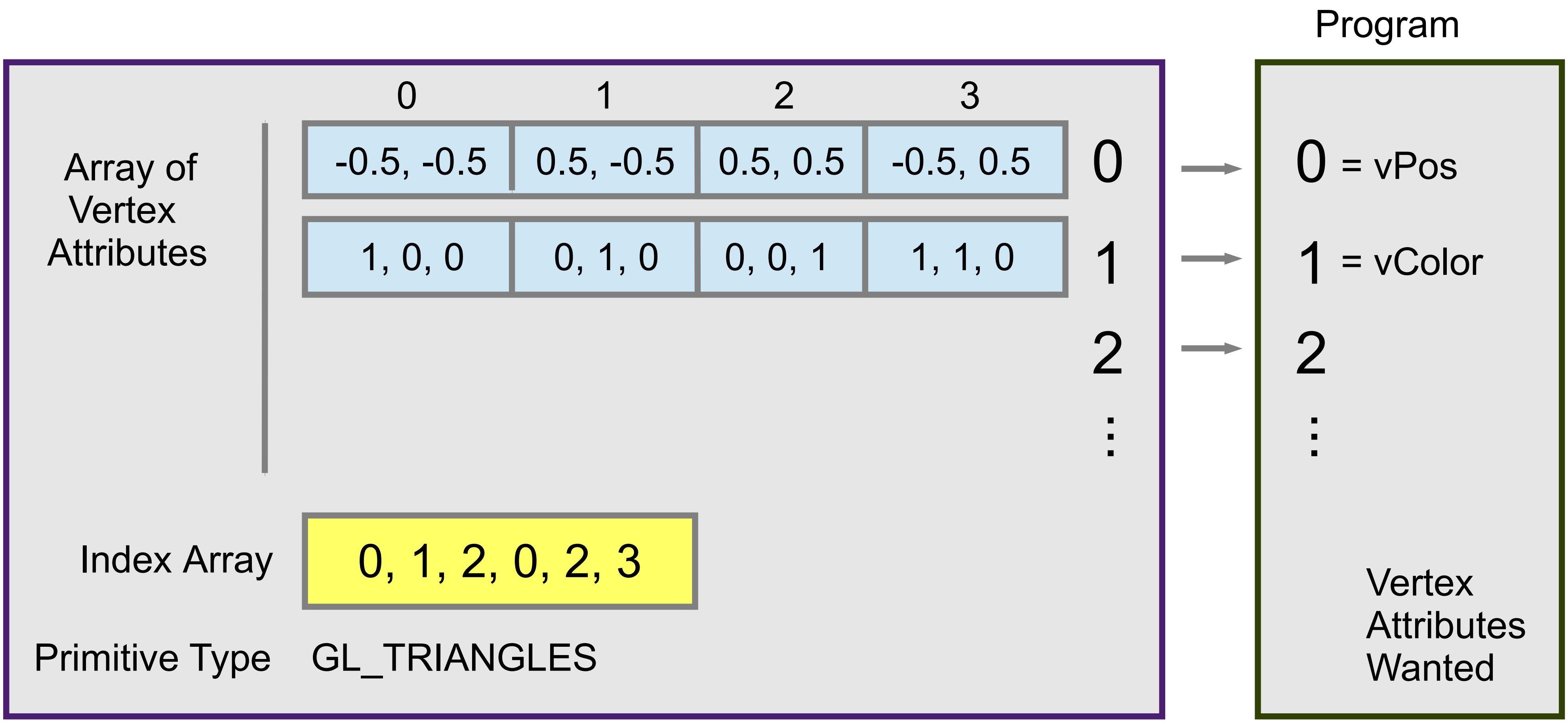
0	1	2	3	
-0.5, -0.5	0.5, -0.5	0.5, 0.5	-0.5, 0.5	0
1, 0, 0	0, 1, 0	0, 0, 1	1, 1, 0	1
				2
				⋮

Index Array 0, 1, 2, 0, 2, 3

Primitive Type GL_TRIANGLES

Attribute Bindings

- Shader program draws vertices using array of attributes
- Program declares a variable (vPos, vColor) for each attribute



Array of Vertex Attributes

	0	1	2	3
0	-0.5, -0.5	0.5, -0.5	0.5, 0.5	-0.5, 0.5
1	1, 0, 0	0, 1, 0	0, 0, 1	1, 1, 0
2				
⋮				

Index Array 0, 1, 2, 0, 2, 3

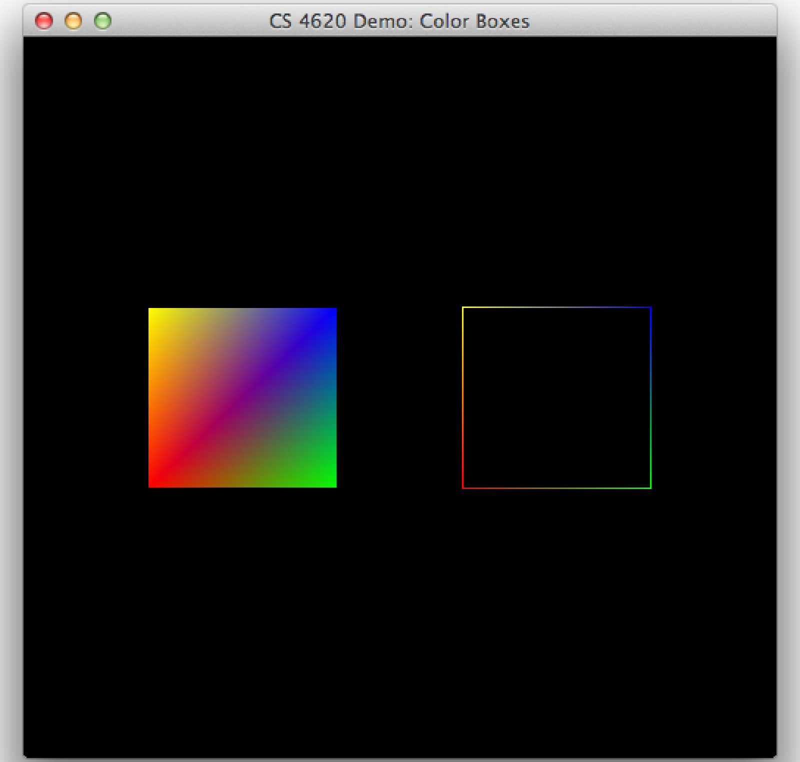
Primitive Type GL_TRIANGLES

Program

0 = vPos
1 = vColor
2
⋮

Vertex Attributes Wanted

Demo: Color Boxes



ColorBoxes' onEntry()

```
@Override
       public void onEntry(GameTime gameTime) {

          ... // fill vertexBuffer as before

          float [] vertexColors = {
                 1.0f, 0.0f, 0.0f,   // vertex 0
                 0.0f, 1.0f, 0.0f,   // vertex 1
                 0.0f, 0.0f, 1.0f,   // vertex 2
                 1.0f, 1.0f, 0.0f    // vertex 3
                    };

          vbColor = GLBuffer.createAsVertex(vertexColors, 3,
                       BufferUsageHint.StaticDraw);
          ...
```

ColorBoxes' draw()

```
...
// Use box vertex positions and colors as we defined before
// Bind attribute array values to a variable in the shader

vb.useAsAttrib(program.getAttribute("vPos"));
vbColor.useAsAttrib(program.getAttribute("vColor"));

// Draw the two boxes as we did before
...
```

Transformations

Representing Transformations

- We will use Matrix3, Matrix4 classes
  - Set matrix contents:

```
Matrix3 translate = new Matrix3(1.0f, 0.0f, 3.0f,
                                0.0f, 1.0f, -4.0f,
                                0.0f, 0.0f, 1.0f);
```

  - Copy matrices:

```
Matrix3 translateAgain = new Matrix3(translate);
```

  - Multiply matrices:

```
translate.mulBefore(translateAgain);
// translate = translate * translateAgain
```

  - Transform points:

```
Vector3 vert = new Vector3(2.0f, 0.0f, 1.0f);
translate.mulPos(vert);  // vert = translate * vert
```

Matrix3/4 Class

- Static functions provide various useful transformation matrices
  - Identity matrices
  - Translations, scales, rotations
  - Projection matrices

Transforming a Vertex

- GLBuffer gives vertices to shader program
- Program transforms them onto screen before drawing

$$\begin{matrix}\text{Screen}\\\text{position}\end{matrix} = \begin{bmatrix} \quad \text{Transformation} \quad \end{bmatrix} \begin{bmatrix} px \\ py \\ pz \\ 1 \end{bmatrix}$$

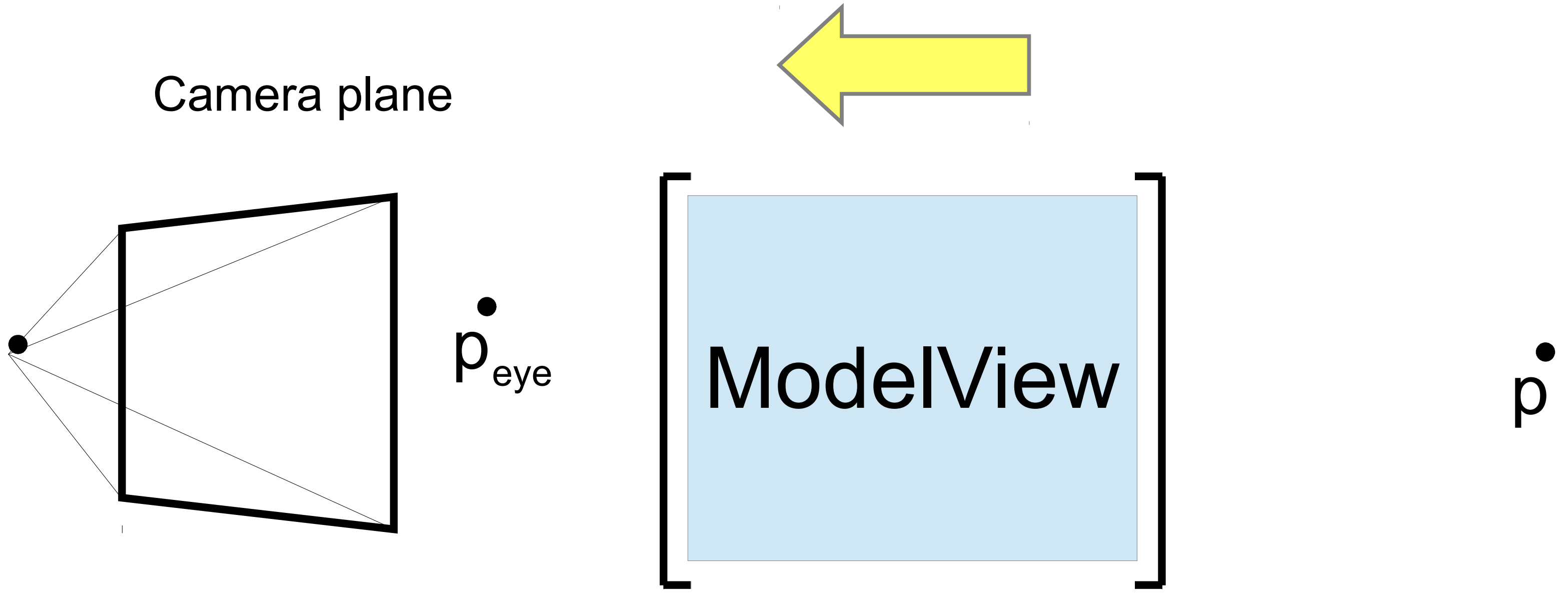
Transforming a Vertex

- Applies two transforms: Projection and ModelView
- Very basic description now (we have previously discussed this topic)

$$\begin{array}{c}\text{Screen}\\\text{position}\end{array} = \begin{bmatrix} \text{Projection} \end{bmatrix} \begin{bmatrix} \text{ModelView} \end{bmatrix} \begin{bmatrix} px \\ py \\ pz \\ 1 \end{bmatrix}$$

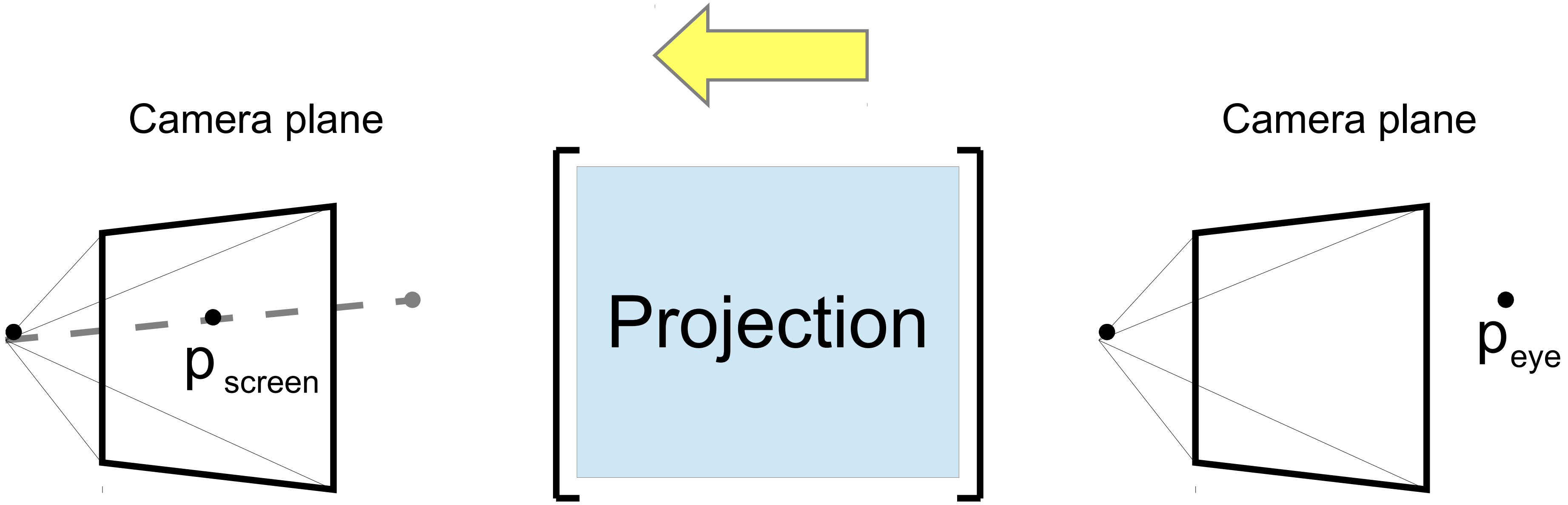
ModelView matrix

- Transforms vertex into coordinates of the viewer
- Modify matrix to transform objects to different places



Projection matrix

- Projects point down onto camera plane
- Usually set matrix once at beginning of draw()

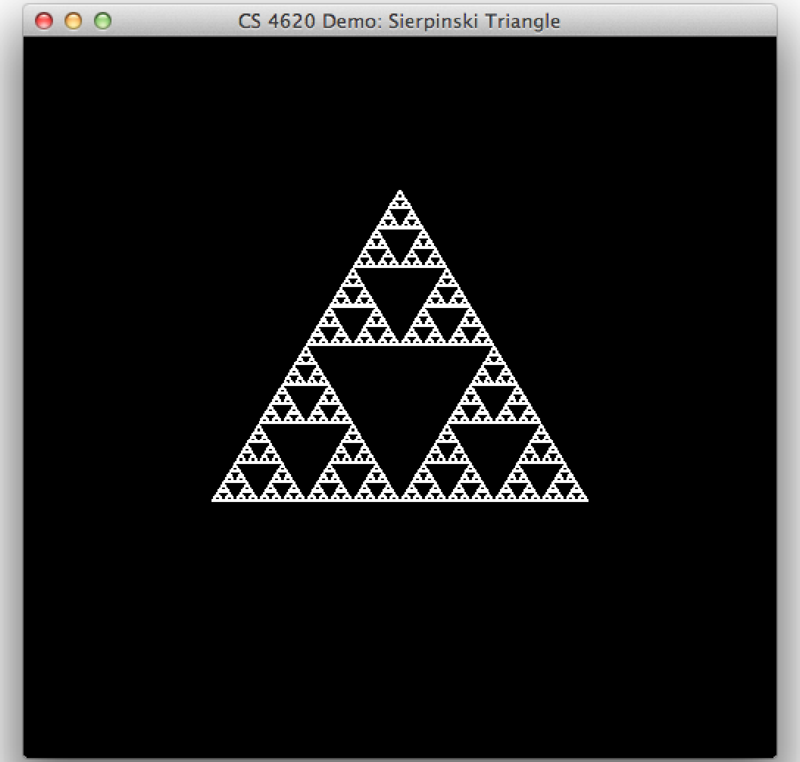


Setting Matrices

- You tell the shader program what matrices to use
  - GLUniform.*setST(program.getUniform("VP"), tr, false);*
  - You can bind a matrix to a variable in the shader, similarly to vertex attributes

- **Program keeps drawing with the same matrices until you change them**
- Common use pattern:
  - GLUniform.*setST(program.getUniform(“tr"),* transformForThisObject*, false);*
  - GL11.glDrawElements(…)
  - Define other transformation
  - Draw something else

Demo: Sierpinski Triangle

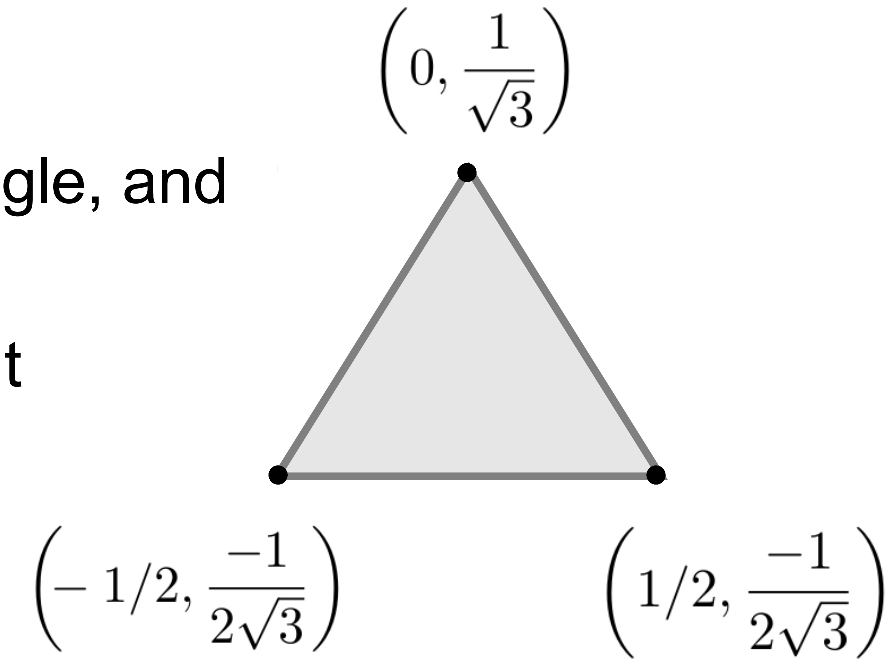


Sierpinski's draw()

```
@Override
public void draw(GameTime gameTime) {
        GL11.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        GL11.glClear(GL11.GL_COLOR_BUFFER_BIT);

        program.use();

        GLUniform.set(program.getUniform("uGridColor"),
        new Vector4(1, 1, 1, 1));

        // Use box vertices
        vb.useAsAttrib(program.getAttribute("vPos"));

        // Transformation
        Matrix4 tr = new Matrix4();
        tr.mulAfter(Matrix4.createTranslation(
        new Vector3(0.0f, -(float)Math.sqrt(3)/6, 0.0f)));

        sierpinski(ibLines, tr, 10);

        GLProgram.unuse();
}
```

Recursively Drawing Fractal

- sierpinski(gl, tr, k):
  - Draw a level-k Sierpinski triangle, and
  - transform by tr
  - k = 0: draw triangle to the right


$\left(0, \frac{1}{\sqrt{3}}\right)$
$\left(-1/2, \frac{-1}{2\sqrt{3}}\right)$
$\left(1/2, \frac{-1}{2\sqrt{3}}\right)$

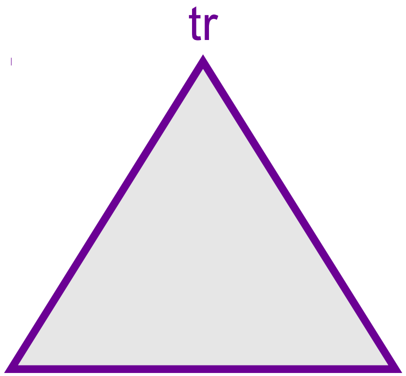
- Recursively: at level k, draw three k-1 Sierpinski triangles, transforming them to the three corners of the triangle

sierpinski()

```
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {
          if (k == 0) {
                    GLUniform.setST(program.getUniform("VP"), tr, false);

                    // Draw the triangle
                    ibLines.bind();
                    GL11.glDrawElements(GL11.GL_LINES, indexCountLines, GLType.UnsignedInt, 0);
                    ibLines.unbind();
          } else {
                    Matrix4 next;

                    //draw the up triangle
                    next = new Matrix4(tr);
                    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
                    next.mulAfter(Matrix4.createTranslation(new Vector3(0.0f,
                    0.5f / (float)Math.sqrt(3.0f), 0.0f)));
                    sierpinski(lines, next, k-1);

                    //draw the right triangle
                    next = new Matrix4(tr);
                    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
                    next.mulAfter(Matrix4.createTranslation(new Vector3(0.25f,
                    -0.25f / (float)Math.sqrt(3.0f), 0.0f)));
                    sierpinski(lines, next, k-1);

                    //draw the left triangle
                    next = new Matrix4(tr);
                    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
                    next.mulAfter(Matrix4.createTranslation(new Vector3(-0.25f,
                    -0.25f / (float)Math.sqrt(3.0f), 0.0f)));
                    sierpinski(lines, next, k-1);
          }
}
```

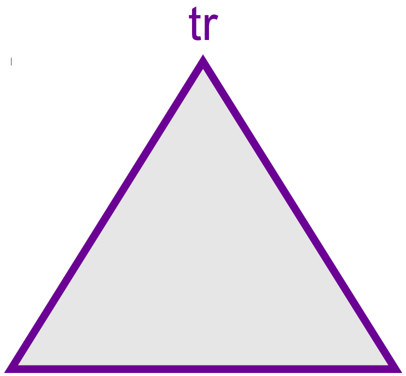
sierpinski()

```
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {
          if (k == 0) {
                    GLUniform.setST(program.getUniform("VP"), tr, false);

                    // Draw the triangle
                    ibLines.bind();
                    GL11.glDrawElements(GL11.GL_LINES,
                    indexCountLines, GLType.UnsignedInt, 0);
                    ibLines.unbind();
          } else {




                    [...]







          }
}
```



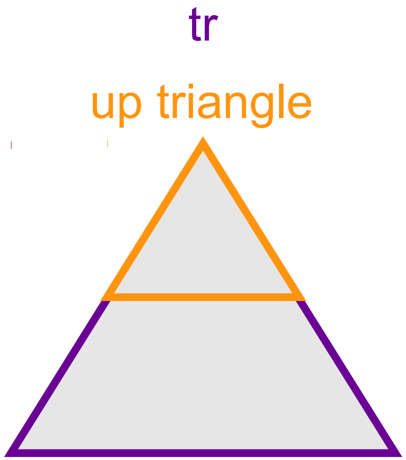
sierpinski()

```
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {

        [...]

        } else {
                Matrix4 next;

                //draw the up triangle
                next = new Matrix4(tr);
                next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
                next.mulAfter(Matrix4.createTranslation(
                new Vector3(0.0f, 0.5f / (float)Math.sqrt(3.0f), 0.0f)));
                sierpinski(lines, next, k-1);




                [...]
        }
}
```



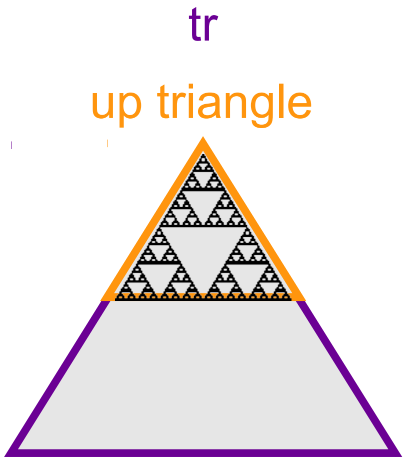
sierpinski()

```
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {

          [...]

          } else {
                    Matrix4 next;

                    //draw the up triangle
                    next = new Matrix4(tr);
                    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
                    next.mulAfter(Matrix4.createTranslation(
                    new Vector3(0.0f, 0.5f / (float)Math.sqrt(3.0f), 0.0f)));
                    sierpinski(lines, next, k-1);






                    [...]
          }
}
```



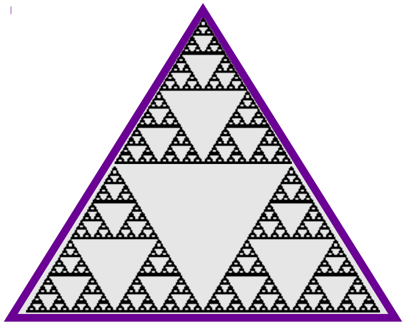
sierpinski()

```
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {

          [...]

          } else {
                    Matrix4 next;

                    //draw the up triangle
                    next = new Matrix4(tr);
                    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
                    next.mulAfter(Matrix4.createTranslation(
                    new Vector3(0.0f, 0.5f / (float)Math.sqrt(3.0f), 0.0f)));
                    sierpinski(lines, next, k-1);

                    [...]
          }
}
```

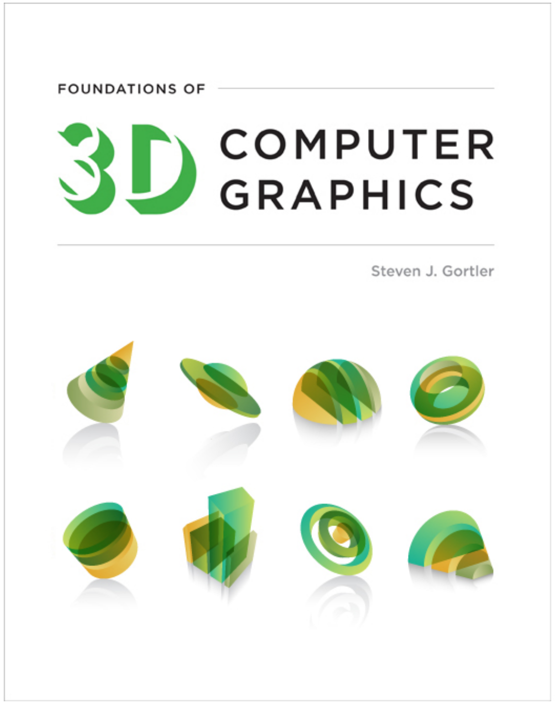


sierpinski()

```
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {
          if (k == 0) {
                    GLUniform.setST(program.getUniform("VP"), tr, false);

                    // Draw the triangle
                    ibLines.bind();
                    GL11.glDrawElements(GL11.GL_LINES, indexCountLines, GLType.UnsignedInt, 0);
                    ibLines.unbind();
          } else {
                    Matrix4 next;

                    //draw the up triangle
                    next = new Matrix4(tr);
                    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
                    next.mulAfter(Matrix4.createTranslation(new Vector3(0.0f,
                    0.5f / (float)Math.sqrt(3.0f), 0.0f)));
                    sierpinski(lines, next, k-1);

                    //draw the right triangle
                    next = new Matrix4(tr);
                    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
                    next.mulAfter(Matrix4.createTranslation(new Vector3(0.25f,
                    -0.25f / (float)Math.sqrt(3.0f), 0.0f)));
                    sierpinski(lines, next, k-1);

                    //draw the left triangle
                    next = new Matrix4(tr);
                    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
                    next.mulAfter(Matrix4.createTranslation(new Vector3(-0.25f,
                    -0.25f / (float)Math.sqrt(3.0f), 0.0f)));
                    sierpinski(lines, next, k-1);
          }
}
```



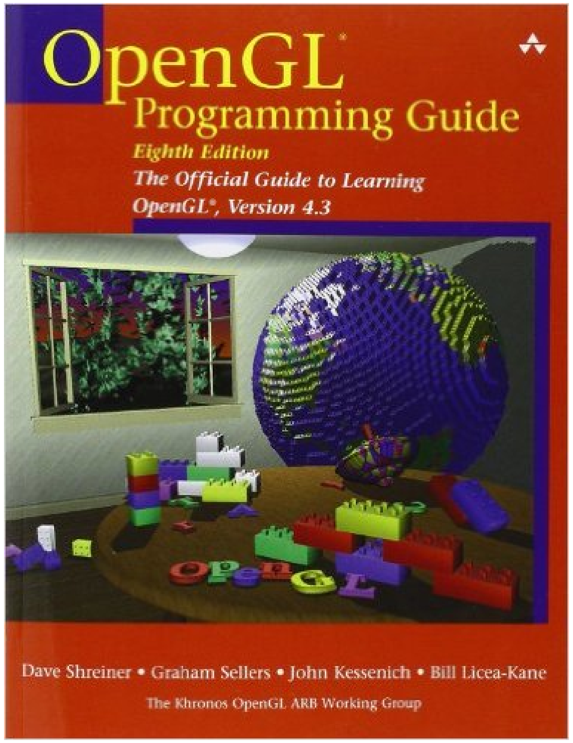
Books and resources



FOUNDATIONS OF

3D COMPUTER GRAPHICS

Steven J. Gortler



OpenGL® Programming Guide

Eighth Edition

The Official Guide to Learning OpenGL®, Version 4.3

Dave Shreiner • Graham Sellers • John Kessenich • Bill Licea-Kane

The Khronos OpenGL ARB Working Group

OpenGL

LWJGL

Lightweight Java Game Library