CS4621/5621 Fall 2015

Computer Graphics Practicum
Intro to OpenGL/GLSL
Part II

Professor: Kavita Bala
Instructor: Nicolas Savva

with slides from Balazs Kovacs, Eston Schweickart, Daniel Schroeder, Jiang Huang and Pramook Khungurn over the years.

Announcements

- Course website:

  http://www.cs.cornell.edu/courses/CS4621/2015fa/
- Piazza for practicum:

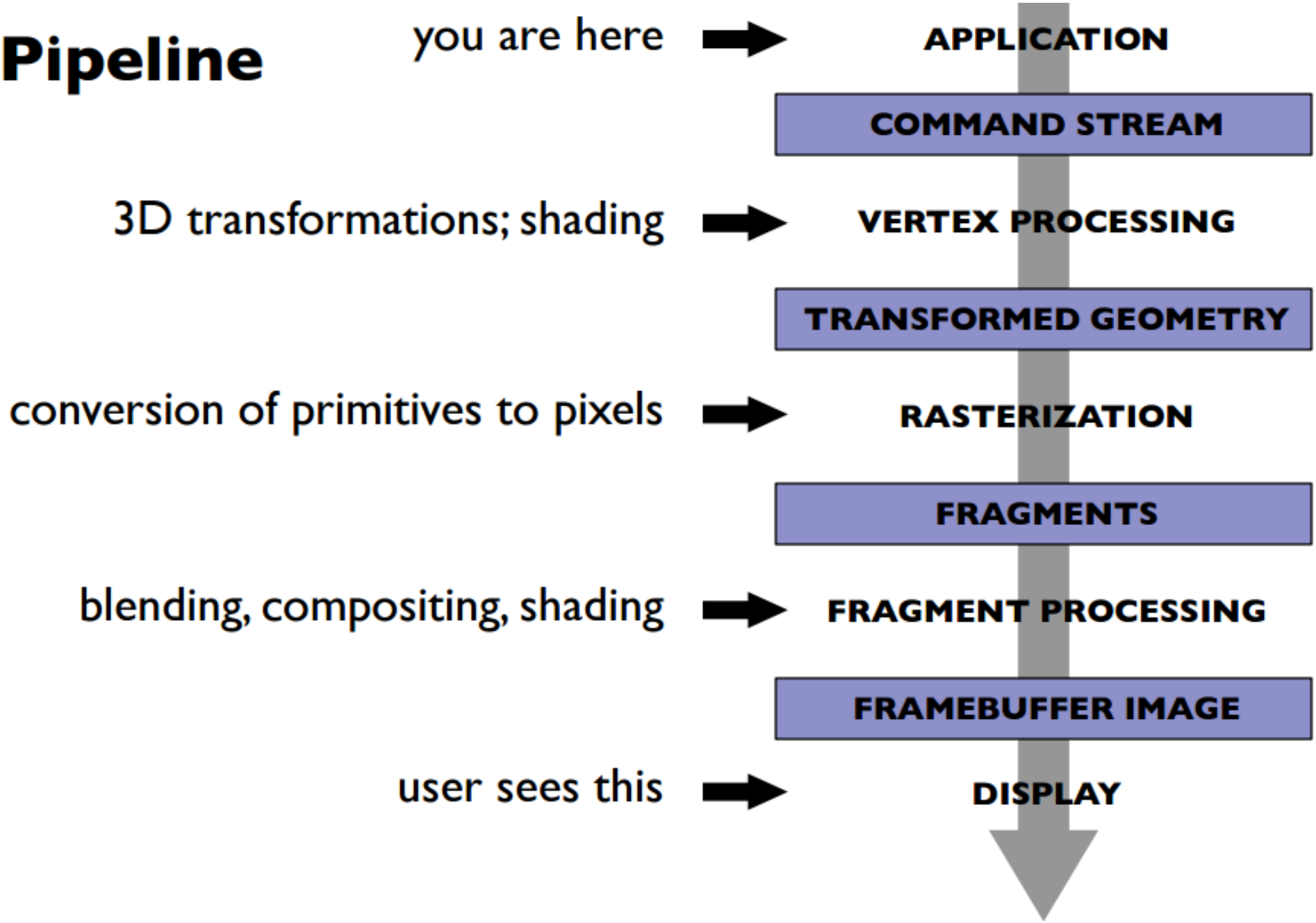
  https://piazza.com/cornell/fall2015/cs46215621/
- OpenGL Diagnostic v2
- First assignment GPUray out today (due Oct 15)

Today

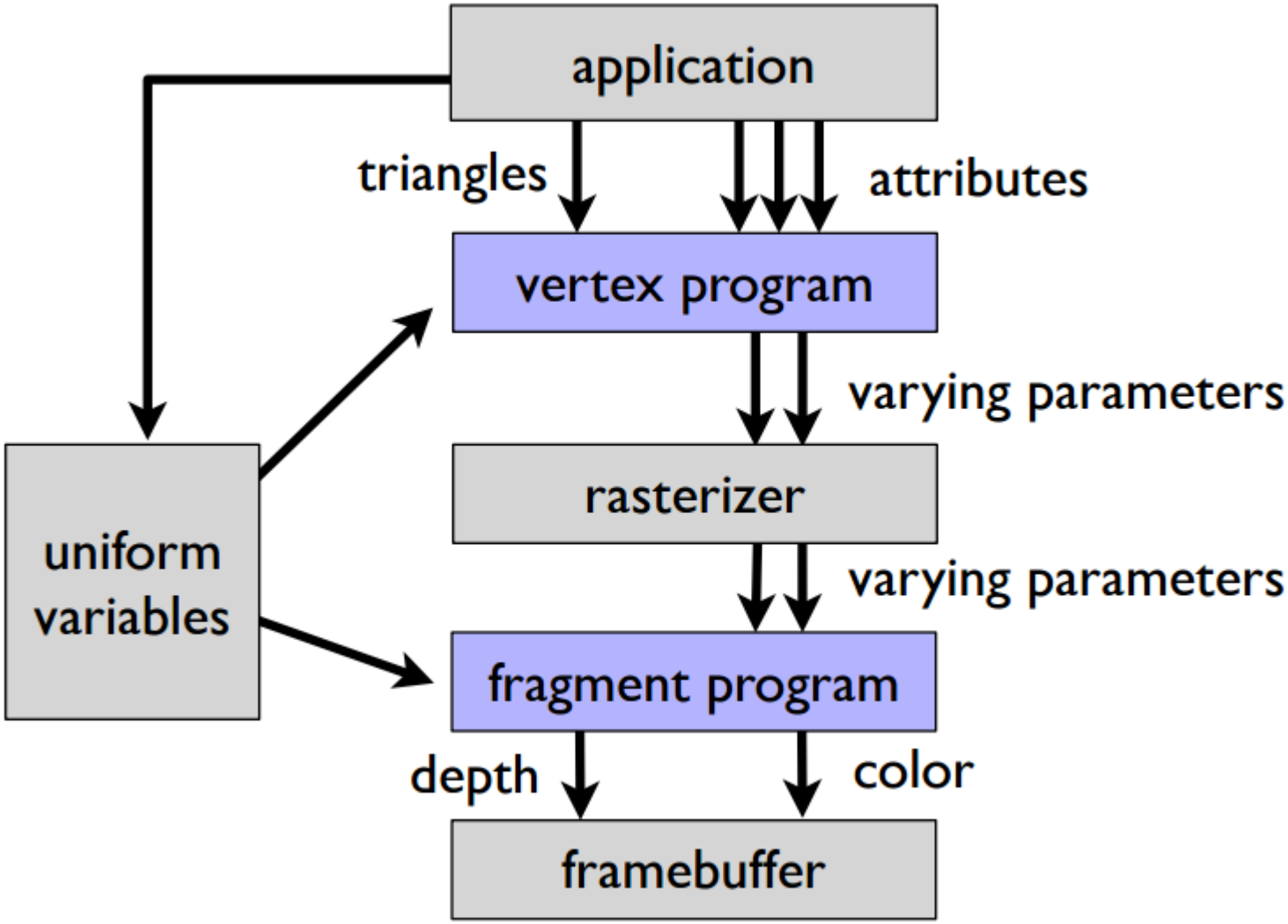
- GPUray
- GLSL basics review
- OpenGL / LWJGL basics

Review

Pipeline


you are here
APPLICATION
COMMAND STREAM
3D transformations; shading
VERTEX PROCESSING
TRANSFORMED GEOMETRY
conversion of primitives to pixels
RASTERIZATION
FRAGMENTS
blending, compositing, shading
FRAGMENT PROCESSING
FRAMEBUFFER IMAGE
user sees this
DISPLAY

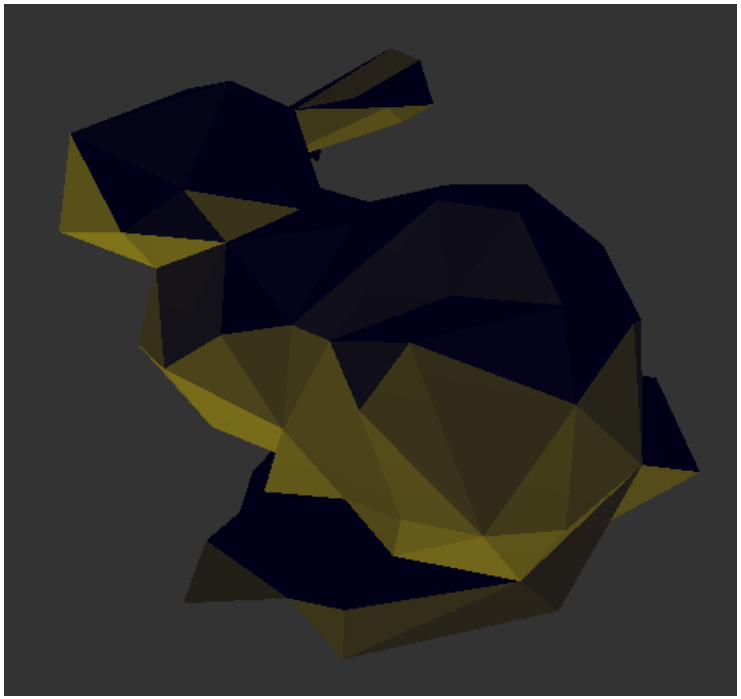
GLSL passing data around



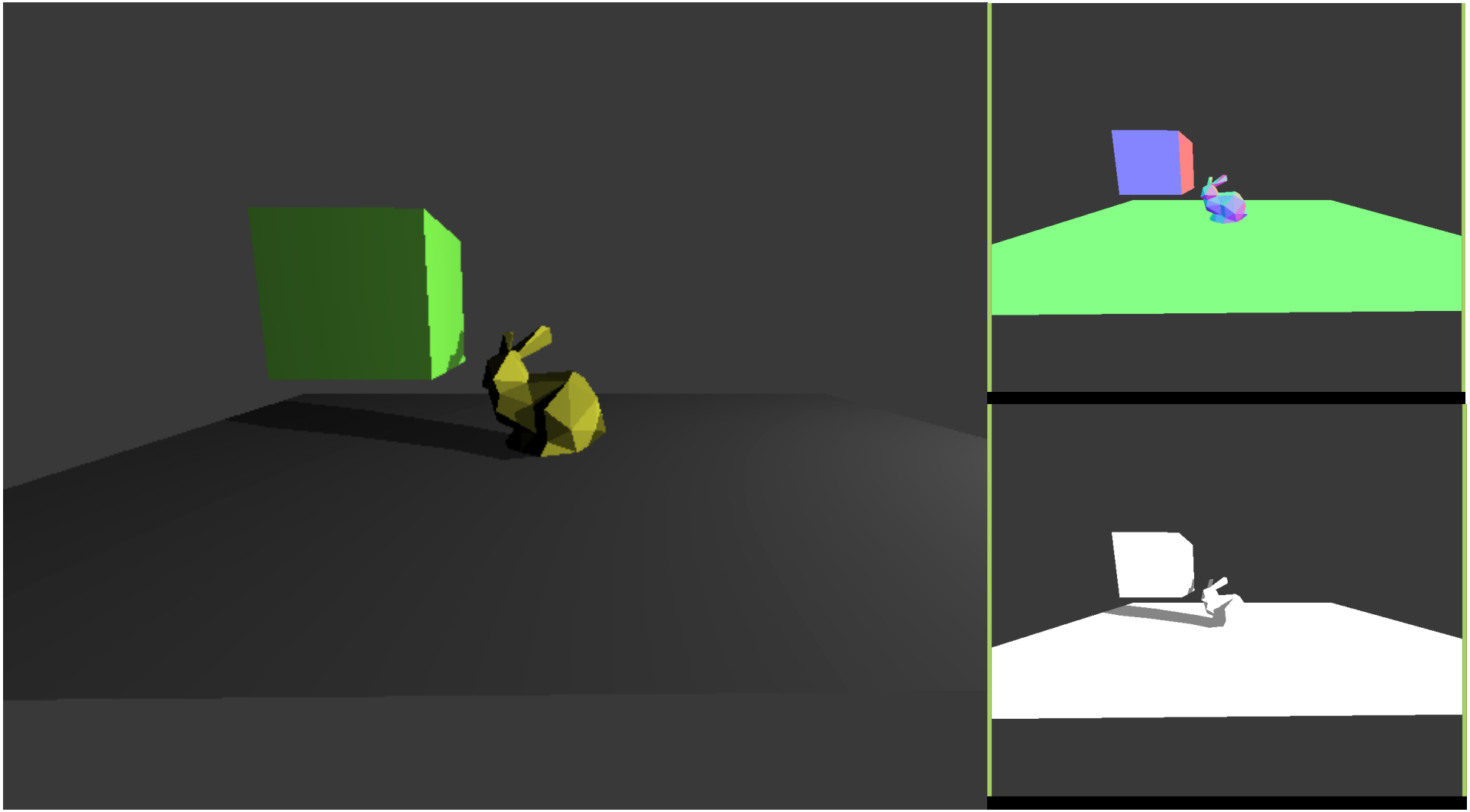
GPUray Overview

- PPA1: port a simplified version of ray1 to GPU
- Mostly deal with the fragment shader
- LWJGL bindings

  (communicating with the GPU)
- Shader compiling and linking
- GLSL code



GPUray Demo



GPUray Tasks

- Setting up the CS4620/4621 framework and OpenGL
- Parsing ray1 xml scenes
- Writing GLSL shader code
- Creating the GLProgram and binding buffers to the GPU
- Adding user interaction
- Performance Benchmark

Creating an OpenGL Program using LWJGL revisited

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

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
}
```

OpenGL Commands and Resources

OpenGL Commands

- Uses 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 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)

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 *sizes);
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 Texture for 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 **GetBufferParameter**[i64]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 **GetNamedBufferParameter**[i64]v(uint buffer, enum pname, int[64]*data);

void **GetBufferPointerv**(enum target, enum pname, void **params);
target: *See BindBuffer*
pname: BUFFER_MAP_POINTER

void **GetNamedBufferPointerv**(uint buffer, enum pname, 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);

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 **GetInteger**v(TIMESTAMP, int *data);

void **GetInteger64v**(TIMESTAMP, int64 *data);

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 **CreateShaderProgram**v(enum type, sizei count, const char * const * strings);

void **ProgramParameter**i(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 **GetProgramInterface**iv(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, (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_(VEC n, MAT n, MAT nxm), DOUBLE, FLOAT_(VEC n, MAT n, MAT nxm), FLOAT, INT, INT_VEC n, (UNSIGNED_INT)_VEC n], BOOL, BOOL_VEC n, 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 ui}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 **ProgramUniformMatrix**{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: TEXTURE i (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{if}*

void **SamplerParameter**{i ui}v(uint sampler, enum pname, const T *params);
pname: *See SamplerParameter{if}*

void **DeleteSamplers**(sizei count,

Sampler Queries [8.3]

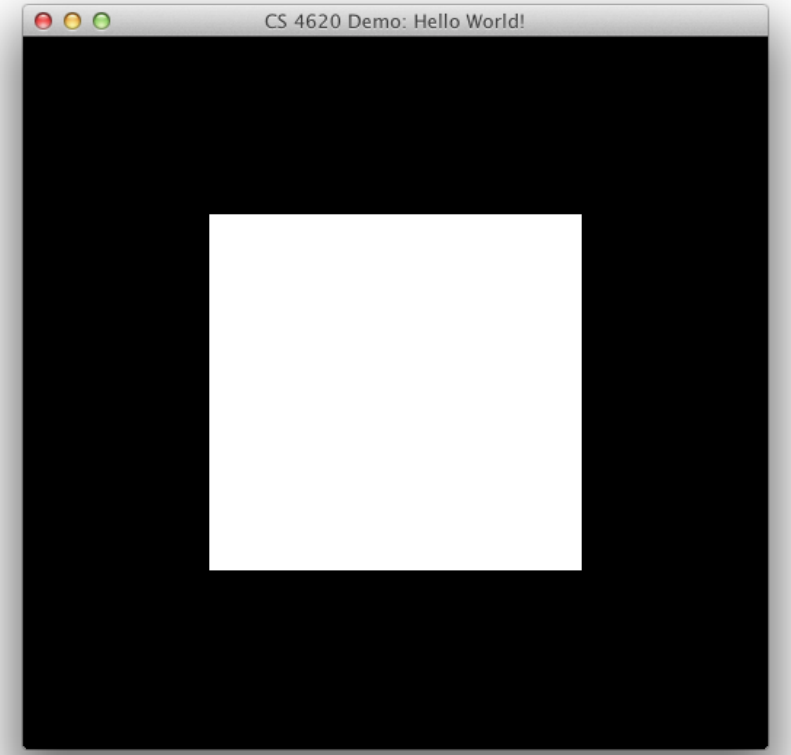
void **GetSamplerParameter**{i f}v(uint sampler, enum pname, T *params);
pname: *See SamplerParameter{if}*

void **GetSamplerParameter**{i ui}v(uint sampler, enum pname, T *params);
pname: *See SamplerParameter{if}*

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,

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 mat 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 that we want to use the vertices in the vertex buffer in a certain way

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

GLSL Program

- Specifies how OpenGL should draw geometry
- Program: A collection of shaders that run together
  - At least one vertex shader or one fragment shader
- At any time, the GPU runs only one program
  - Must specify program to use before drawing geometry

To use a GLSL program…

Follow the next 7 steps:

1. Create shader objects.
2. Read source code from files and feed them to the shader objects just created.
3. Compile the shader.
4. Create a program object.
5. Attach the shaders to the program.
6. Link the program.
7. Tell OpenGL to use your shader program.

Now, to create a GLSL program…

- Create a GLProgram object

```
private GLProgram program;

public void onEntry(GameTime gameTime) {

  program = new GLProgram();
  program.quickCreateResource(
      "cs4621/demos/shaders/Grid.vert",  // Path to vertex shader
      "cs4621/demos/shaders/Grid.frag",  // Path to fragment shader
      null);                             // Optional attribute list
}
```

CS 4620/4621 Framework

- Contains GLProgram class to abstract OpenGL calls:
  - Added convenience methods
  - Help keep conventions straight
  - Controls mapping between attribute variables and vertex buffers

Back to GLSL

OpenGL/GLSL Plumbing

- Suppose we have already created the program
- We tell OpenGL to use it.
- We then instruct OpenGL to draw the two triangles:

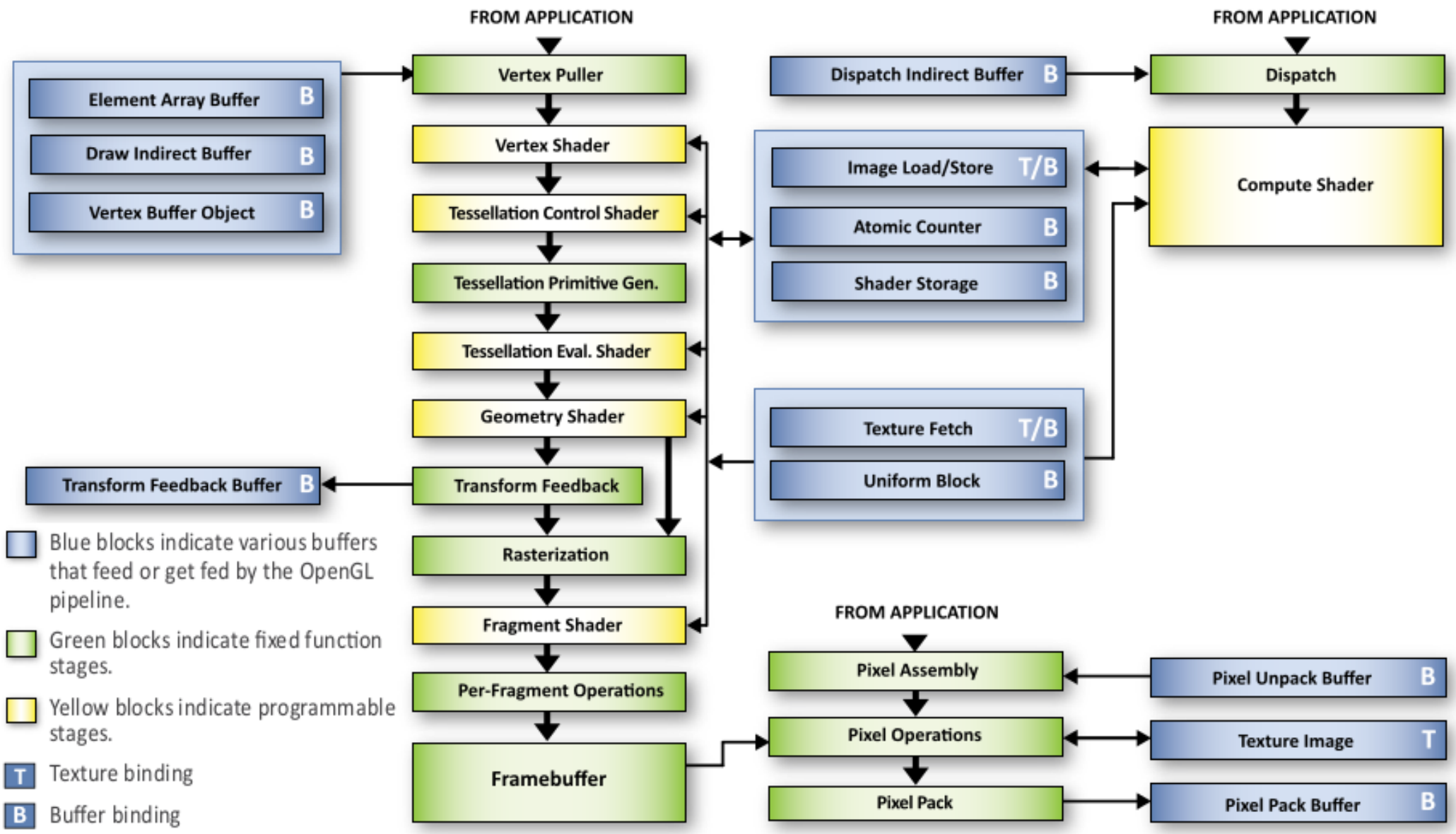
HelloWorldScreen.java:

```
// The vertices in our vertex buffer, initialized earlier
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
                   };
//...
// In the draw method
program.use();

glDrawElements(...);

GLProgram.unuse();
```

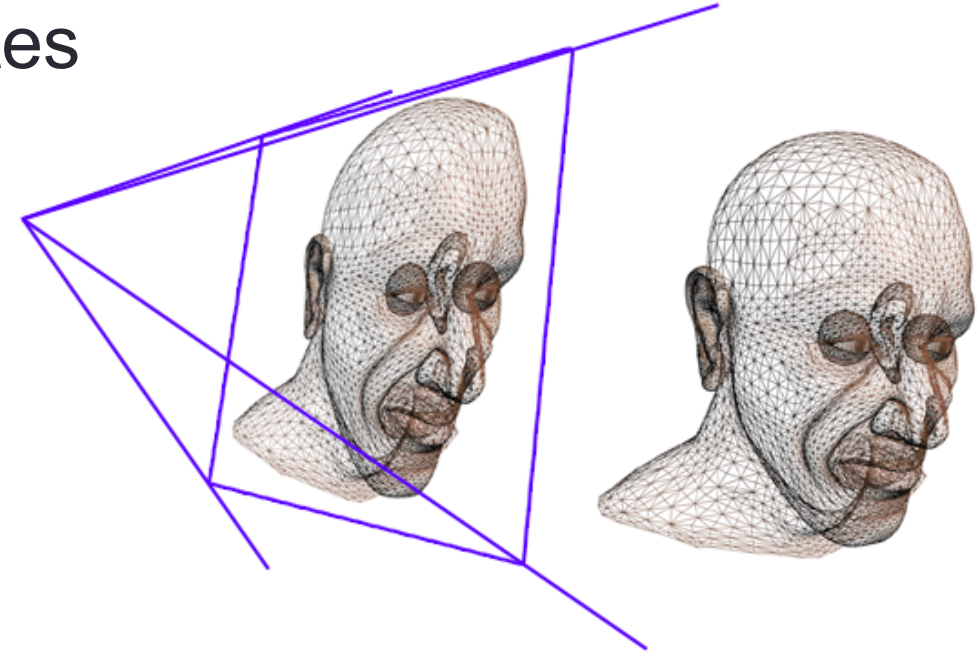
Modern Graphics Pipeline



Vertex Shader

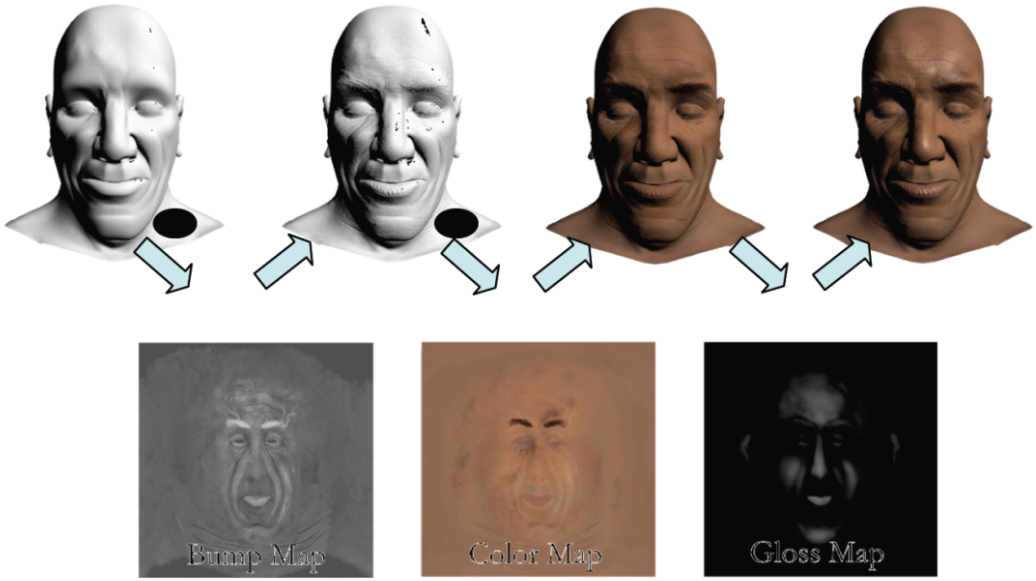
- Transform vertices from object space to clip space

- Compute other data that are interpolated with vertices
  - Color
  - Normals
  - Texture coordinates
  - Etc.

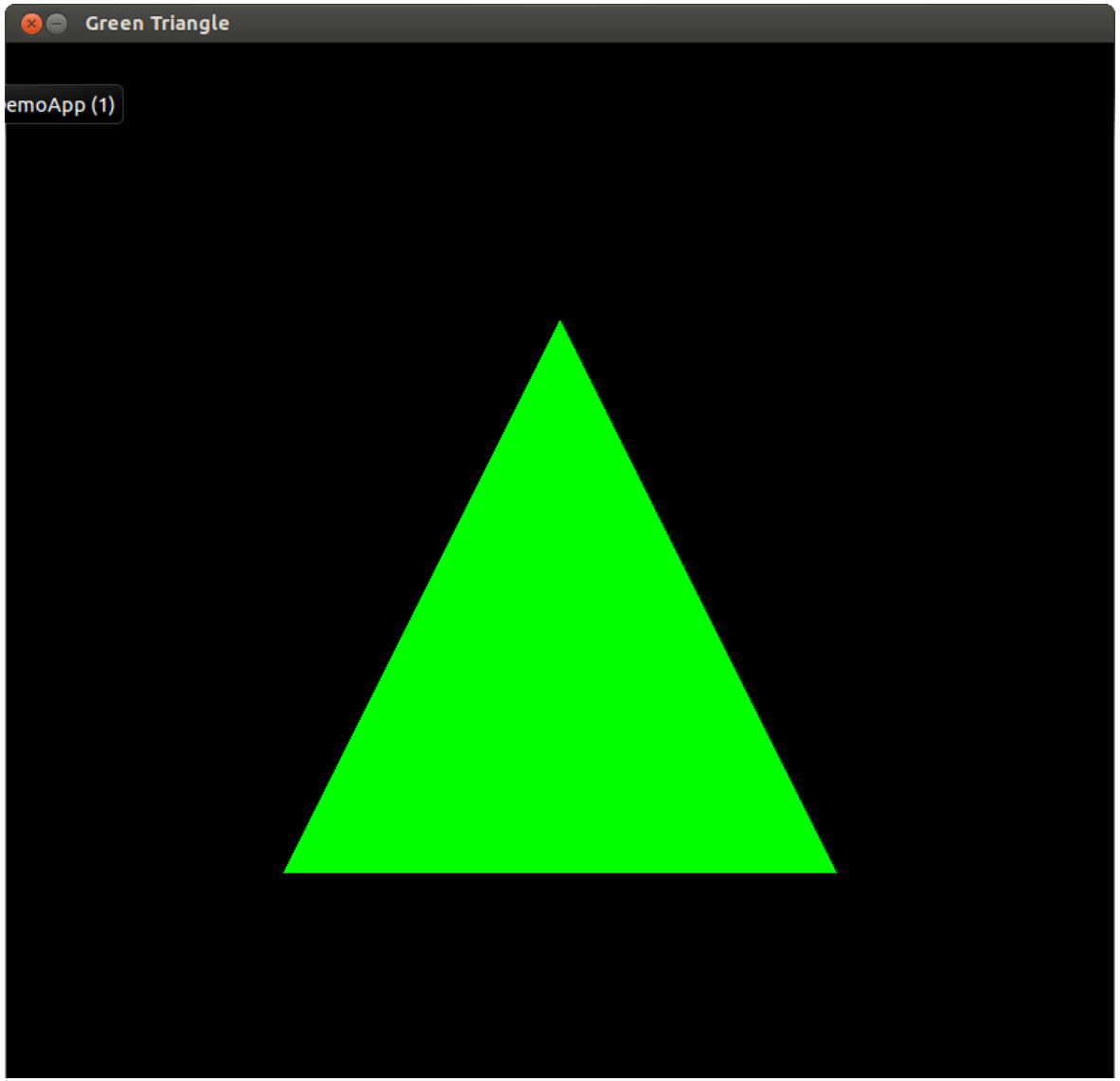


Fragment Shaders

- Compute the color of a fragment (i.e. a pixel)
- Take interpolated data from vertex shaders
- Can read more data from:
  - Textures
  - User specified values



Green Triangle Demo



Vertex Shader

green_shader.vert:

```
#version 120

uniform mat4 VP;
attribute vec3 vPos;

void main()
{
    gl_Position = VP * vec4(vPos,1);
}
```

Fragment Shader

green_shader.frag:

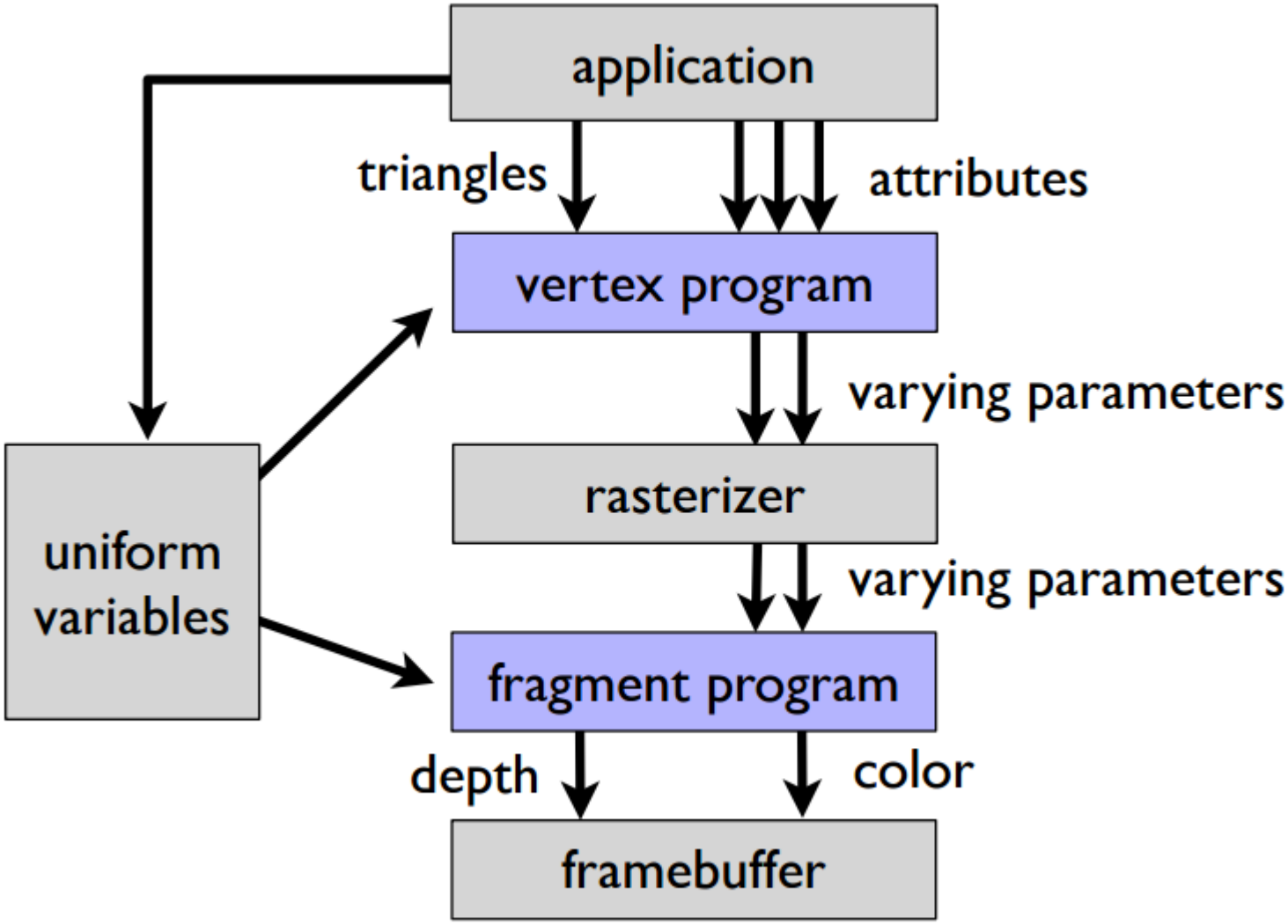
```
#version 120

void main()
{
  gl_FragColor = vec4(0,1,0,1);
}
```

GLSL Data Types

- Both in GLSL and Java
  - float, int
- GLSL has, but Java does not have
  - vec2, vec3, vec4: vectors
  - mat2, mat3, mat4: matrices
  - sampler1D, sampler2D, sample3D, samplerCube, etc: textures
- Java has, but GLSL does not have
  - Object
  - String
  - etc...

GLSL passing data around



Uniform Variable

- A GLSL variable the user can specify value from the C/Java side.
- Its value is constant while drawing each vertex and pixel.
- Suitable for specifying
  - Material properties
  - Transformation matrices
  - Light sources
  - Textures

Declaring a Uniform Variable in GLSL

- Declare as a global variable (outside functions).
- Prefix the variable type with keyword “uniform.”
- Examples:

```
// The values for these are initialized in the Java code!
uniform float shininess;
uniform vec3 color;
uniform mat4 model_transform;

void main()
{
    // Code here...
}
```

Caveats

- Uniform variables are shared between vertex and fragment shaders
  - Declare once in vertex shader and once more in fragment shader.
- As a result, types of uniform variables in vertex and fragment shaders must be consistent.
- Cannot have **uniform int x**; in vertex shader, but **uniform float x;** in fragment shader.
- Uniforms that are declared but not used are “optimized” out
  - OpenGL throws an error if you try to set a nonexistent uniform

Using Uniform Variables in the CS4620/4621 Framework

- Uniform variables are encapsulated by `GLUniform` class.
- Use `program.getUniform(<name>)` to get the instance (an integer) corresponding to the name.
- Set values by `GLUniform.set**(...)` methods.

```
// In GLSL: uniform vec3 color;
program.use();
Vector3 c = new Vector3(1.0f, 0.5f, 1.0f);
GLUniform.set(program.getUniform("uGridColor"), c);

// In GLSL: uniform mat4 MVP;
// For matrices, use setST, not set! A boolean is provided
// for transposing.
GLUniform.setST(program.getUniform("VP"),
    camera.mViewProjection, false);
```

Attribute Variables

- A variable containing an attribute for a single vertex.
- Position, normal, texture coordinate, etc.
- Each time the shader is run, the attribute variables receive the values for the current vertex.
- These only appear in vertex shaders (Why?)

Attribute Mapping

- Attribute variables map to OpenGL buffers.
- OpenGL buffers have an index, GLSL attribute variables have a name.
- Must ensure the mapping from buffer indices to variable names is correct.
- In the provided framework:

```
 // Create a data buffer to fill in the attribute data
 GLBuffer vertexPositions = new
GLBuffer(BufferTarget.ArrayBuffer,   BufferUsageHint.StaticDraw,
true);

 vertexPositions.setAsVertexVec3();

 // Set vertexPositions, e.g. by reading in a Mesh

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

Demo: Twisting

2D Twisting

- We transform vertices according to the following equation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\left(t\sqrt{x^2+y^2}\right) & -\sin\left(t\sqrt{x^2+y^2}\right) \\ \sin\left(t\sqrt{x^2+y^2}\right) & \cos\left(t\sqrt{x^2+y^2}\right) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

where

- (x,y) is the vertex position in object space.
- (x',y') is the vertex position in clip space.
- t is the twisting factor,
  which is stored in the uniform variable “twisting”

TwistScreen (legacy commands)

```
public void draw(GameTime gameTime) {
...
program.use();

   int count = 100; //200;
   float size = 1.0f / count;
   GL11.glBegin(GL11.GL_QUADS);//GL11.GL_LINES
   //GL11.glBegin(GL11.GL_POINTS);

    for(int i=0;i<count;i++)
        for(int j=0;j<count;j++)
        {
            float x = -0.5f + i * size;
            float y = -0.5f + j * size;

            GL11.glVertex2f(x,     y);
            GL11.glVertex2f(x+size, y);
            GL11.glVertex2f(x+size, y+size);
            GL11.glVertex2f(x,      y+size);
        }
    GL11.glEnd();
  GLUniform.set(program.getUniform("color"), new Vector3(1, 0, 1));
  GL20.glUniform1f(program.getUniform("twisting"), 0);//(float)gameTime.total);

program.unuse();
}
```

TwistScreen (modern OpenGL calls)

```
public void draw(GameTime gameTime) {
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
   glClear(GL2.GL_COLOR_BUFFER_BIT);

   program.use();

   // Set the uniforms
   GLUniform.set(program.getUniform("color"), color);
   GLUniform.set(program.getUniform("twisting"), 0.5f);

   // Set the attribute

vertexPositions.useAsAttrib(program.getAttribute("in_Vertex"));

   glDrawElements(...); // Draw the mesh

   GLProgram.unuse();
}
```

Vertex Shader Code

```
#version 120

uniform float twisting;

void main()
{
    float angle = twisting * length(gl_Vertex.xy);
    float s = sin(angle);
    float c = cos(angle);
    gl_Position.x = c * gl_Vertex.x - s * gl_Vertex.y;
    gl_Position.y = s * gl_Vertex.x + c * gl_Vertex.y;
    gl_Position.z = 0.0;
    gl_Position.w = 1.0;
}
```

Fragment Shader Code

```
#version 120


uniform vec3 color;

void main()
{
    gl_FragColor = vec4(color, 1);
}
```

Varying Variables

- Interface between vertex and fragment shaders.
- Vertex shader outputs a value at each vertex, writing it to this variable.
- Fragment shader reads a value from the same variable, automatically interpolated to that fragment.
- No need to declare these in the Java program (Why?)

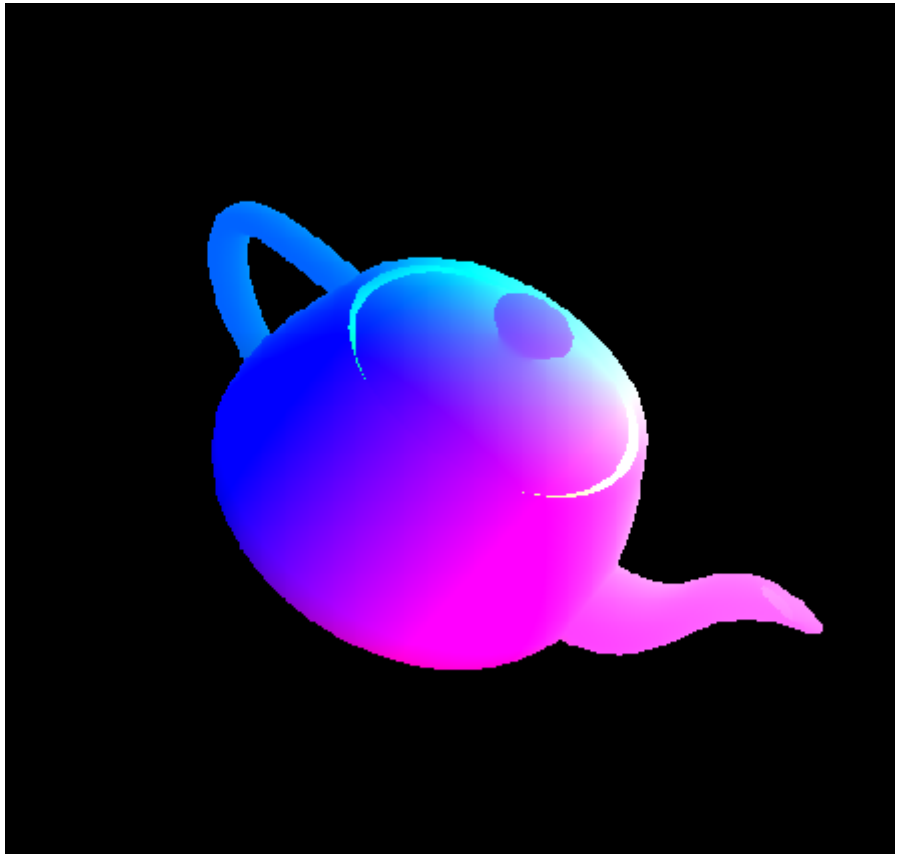
Declaring Varying Variables

- Declare as a global variable (outside functions).
- Syntax: varying <<type>> <<name>>;
- Example:

```
varying vec3 color;

void main()
{
    // Some code here...
}
```

Demo: Position as Color



Position as Color

- Compute the color of each fragment from its position in object space.

$$\text{color} = (\text{position} + (1,1,1)) / 2$$

Vertex Shader Code

```
#version 120

varying vec3 color;
attribute vec3 vPos;
uniform mat4 VP;

void main()
{
   gl_Position = VP * vec4(vPos,1);
   //color = (vec3(vPos) + vec3(1,1,1)) * 0.5;
   color = (vec3(gl_Position.xyz) + vec3(1,1,1)) * 0.5;
}
```

Fragment Shader Code

```
#version 120

varying vec3 color;

void main()
{
   gl_FragColor = vec4(color, 1);
}
```

Vertex Shader Code

```
#version 330


in vec3 vPos;

uniform mat4 VP;

out vec3 color;
out vec4 vPosition;

void main()
{
   vPosition = VP * vec4(vPos,1);
   color = (vec3(gl_Position.xyz) + vec3(1,1,1)) * 0.5;
}
```

Fragment Shader Code

```
#version 330

in vec3 color;
out vec3 vFragColor;

void main()
{
   vFragColor = vec4(color, 1);
}
```