VGP351 – Week 1

- Agenda:
  - Course road-map
  - High-level graphics API overview
    - OpenGL
    - SDL
  - Graphics pipeline introduction
  - Shading language introduction
  - “Hello, world!”

6-April-2010

What should you already know?

- C++ and object oriented programming
  - For most assignments you will need to implement classes or portions of classes that conform to specific interfaces

What should you already know?

- C++ and object oriented programming
  - For most assignments you will need to implement classes or portions of classes that conform to specific interfaces
- Graphics terminology and concepts
  - Polygon, pixel, texture, infinite light, point light, spot light, etc.

What should you already know?

- C++ and object oriented programming
  - For most assignments you will need to implement classes or portions of classes that conform to specific interfaces
- Graphics terminology and concepts
  - Polygon, pixel, texture, infinite light, point light, spot light, etc.
- Linear algebra and vector math
  - Matrix arithmetic

What will you learn?

- Create and use a window for OpenGL drawing
  - As a *cross-platform* graphics interface, OpenGL has no knowledge of windows, mice, keyboards, etc.

What will you learn?

- Create and use a window for OpenGL drawing
  - As a *cross-platform* graphics interface, OpenGL has no knowledge of windows, mice, keyboards, etc.
- Draw static and animated models
  - We'll use the OpenGL Shading Language (GLSL)

What will you learn?

- Create and use a window for OpenGL drawing
  - As a *cross-platform* graphics interface, OpenGL has no knowledge of windows, mice, keyboards, etc.
- Draw static and animated models
  - We'll use the OpenGL Shading Language (GLSL)
- Basic techniques for lighting and shading
  - Shading: flat vs. Gouraud vs. Phong
  - Lighting: Lambertian vs. Phong vs. Blinn

What will you learn?

- Create and use a window for OpenGL drawing
  - As a *cross-platform* graphics interface, OpenGL has no knowledge of windows, mice, keyboards, etc.
- Draw static and animated models
  - We'll use the OpenGL Shading Language (GLSL)
- Basic techniques for lighting and shading
  - Shading: flat vs. Gouraud vs. Phong
  - Lighting: Lambertian vs. Phong vs. Blinn
- Texture mapping

What will we not cover?

- “Fixed function” operations
  - Basically, anything not included in OpenGL ES 2.x
  - The only relevant devices today that do *not* support programmable shaders are...

What will we not cover?

- “Fixed function” operations
  - Basically, anything not included in OpenGL ES 2.x
  - The only relevant devices today that do *not* support programmable shaders are the iPhone 3G and the G1
    - The iPhone 3Gs supports OpenGL ES 2.0 and ES 1.1

What will we not cover?

- “Fixed function” operations
  - Basically, anything not included in OpenGL ES 2.x
  - The only relevant devices today that do *not* support programmable shaders are the iPhone 3G and the G1
    - The iPhone 3Gs supports OpenGL ES 2.0 and ES 1.1
- Advanced lighting and animation techniques
  - That's VGP352

What will we not cover?

- “Fixed function” operations
  - Basically, anything not included in OpenGL ES 2.x
  - The only relevant devices today that do *not* support programmable shaders are the iPhone 3G and the G1
    - The iPhone 3Gs supports OpenGL ES 2.0 and ES 1.1
- Advanced lighting and animation techniques
  - That's VGP352
- Shadows
  - That's VGP353

How will you be graded?

- Four bi-weekly quizzes
  - These are listed on the syllabus
- One final exam
- Five *graded* programming projects
  - Something is due every week
  - Half of these will just be checked
  - Half of these will actually be graded

How will programs be graded?

- Does the program produce the correct output?
- Are appropriate algorithms and data-structures used?
- Is the code readable, clear, and properly documented?

How will programs be graded?

```
long h[4];t(){h[3]-=h[3]/3000;setitimer(0,h,0);}c,d,l,v[]={(int)t,0,2},w,s,I,K
=0,i=276,j,k,q[276],Q[276],*n=q,*m,x=17,f[]={7,-13,-12,1,8,-11,-12,-1,9,-1,1,
12,3,-13,-12,-1,12,-1,11,1,15,-1,13,1,18,-1,1,2,0,-12,-1,11,1,-12,1,13,10,-12,
1,12,11,-12,-1,1,2,-12,-1,12,13,-12,12,13,14,-11,-1,1,4,-13,-12,12,16,-11,-12,
12,17,-13,1,-1,5,-12,12,11,6,-12,12,24};u(){for(i=11;++i<264;)if((k=q[i])-Q[i]
){Q[i]=k;if(i-++I||i%12<1)printf("\033[%d;%dH",(I=i)/12,i%12*2+28);printf(
"\033[%dm  "+(K-k?0:5),k);K=k;}Q[263]=c=getchar();}G(b){for(i=4;i--;)if(q[i?b+
n[i]:b])return 0;return 1;}g(b){for(i=4;i--;q[i?x+n[i]:x]=b);}main(C,V,a)char*
*V,*a;{h[3]=1000000/(l=C>1?atoi(V[1]):2);for(a=C>2?V[2]:"jkl pq";i;i--)*n++=i<
25||i%12<2?7:0;srand(getpid());system("stty cbreak -echo stop u");sigvec(14,v,
0);t();puts("\033[H\033[J");for(n=f+rand()%7*4;;g(7),u(),g(0)){if(c<0){if(G(x+
12))x+=12;else{g(7);++w;for(j=0;j<252;j=12*(j/12+1))for(;q[++j];)if(j%12==10){
for(;j%12;q[j--]=0);u();for(;--j;q[j+12]=q[j]);u();}n=f+rand()%7*4;G(x=17)||(c
=a[5]);}}if(c==*a)G(--x)||++x;if(c==a[1])n=f+4**(m=n),G(x)||(n=m);if(c==a[2])G
(++x)||--x;if(c==a[3])for(;G(x+12);++w)x+=12;if(c==a[4]||c==a[5]){s=sigblock(
8192);printf("\033[H\033[J\033[0m%d\n",w);if(c==a[5])break;for(j=264;j--;Q[j]=
0);while(getchar()-a[4]);puts("\033[H\033[J\033[7m");sigsetmask(s);}}d=popen(
"stty -cbreak echo stop \023;sort -mnr -o HI - HI;cat HI","w");fprintf(d,
"%4d from level %1d by %s\n",w,l,getlogin());pclose(d);}
```

[1] From http://homepages.cwi.nl/~tromp/tetris.html

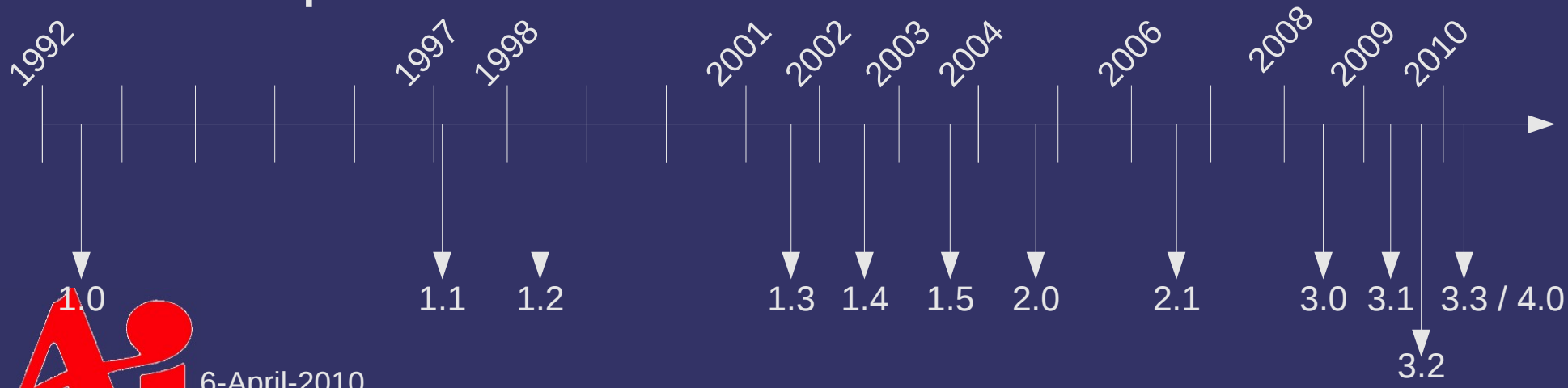
6-April-2010

Class Web Site

- Syllabus, assignments, and base code:
  http://people.freedesktop.org/~idr/2010Q2-VGP351/

10,000 Foot OpenGL Overview

- Created by SGI due to industry demand for a standard more open than Iris GL
  - Originally controlled by the OpenGL Architecture Review Board (ARB)
  - Now controlled by the Khronos Group
- Member companies create and *vote* on changes to the specification



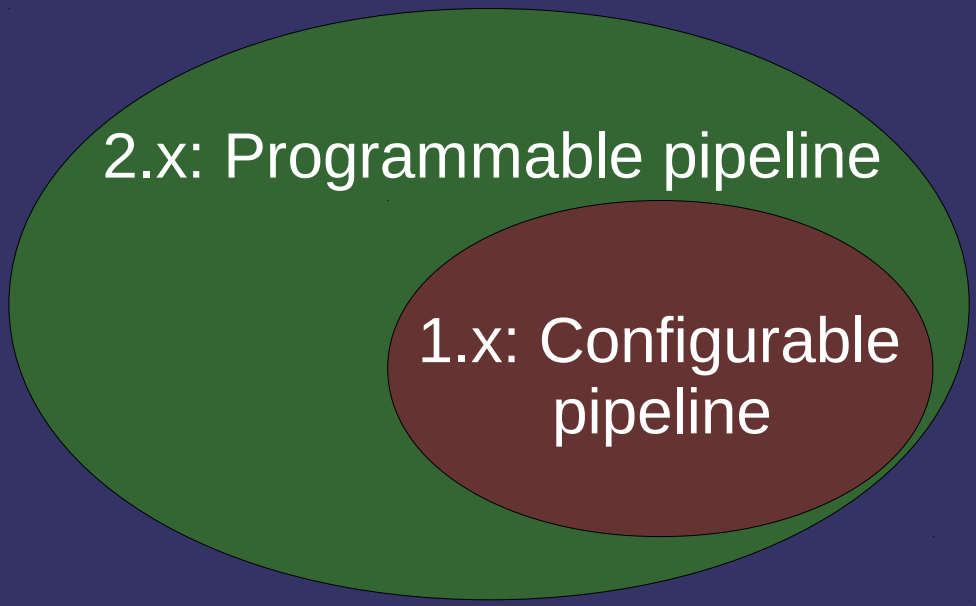
6-April-2010

OpenGL Versions

1.x: Configurable pipeline

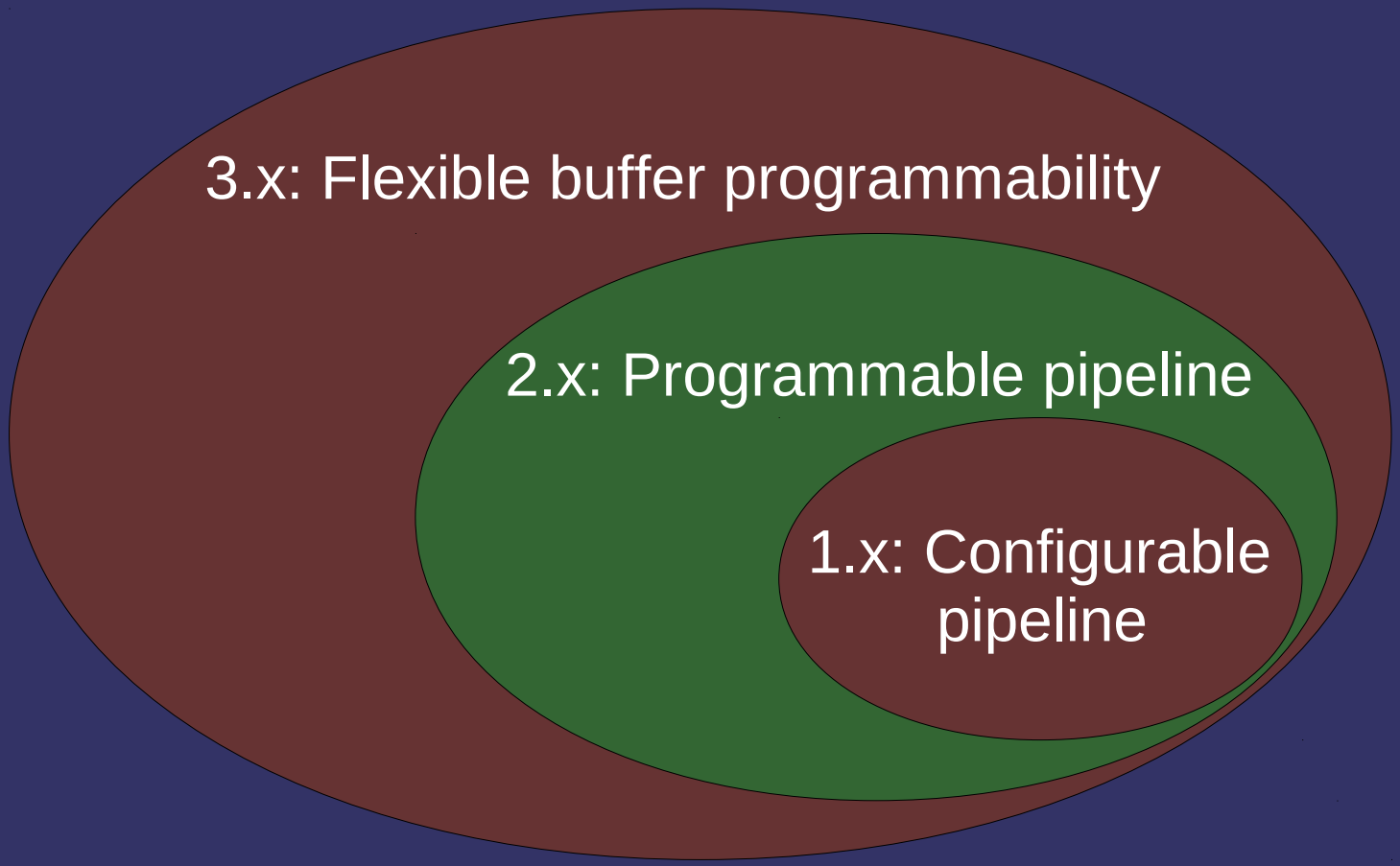
6-April-2010

OpenGL Versions



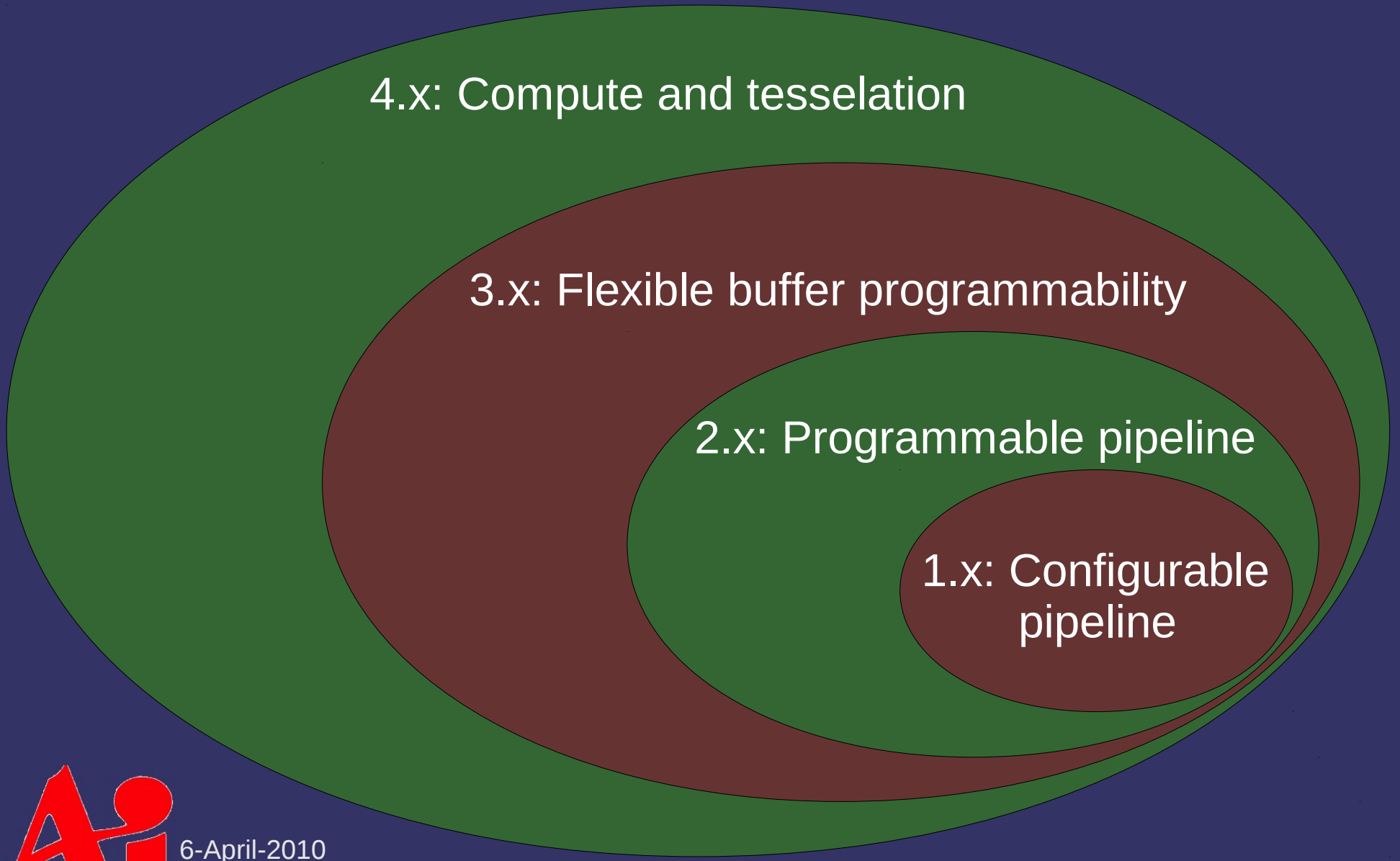
6-April-2010

OpenGL Versions



6-April-2010

OpenGL Versions



6-April-2010

OpenGL Design Principles

- OpenGL is a *low-level*, device independent, platform independent graphics hardware interface
- From *The Design of the OpenGL Graphics Interface*, by Mark Segal and Kurt Akeley:

  "An essential goal of OpenGL is to provide device independence while still allowing complete access to hardware functionality. The API therefore provides access to graphics operations at the lowest possible level that still provides device independence."

OpenGL Design Principles (cont.)

- Based on a client-server model
  - Shows its Unix / X-Windows origins
    - Client (application program) and server (rendering program) were running on different computers
  - Still works!
    - Client (application program) and server (firmware on the gfx card) *are* different computers

OpenGL Design Principles (cont.)

- The GL is a *state machine* with a *push model*
  - Clients send commands that change server state
    - At any time the current state determines what / how objects are rendered
  - Clients send data to the server for rendering
    - Very rarely does data come back from the server
    - So-called “round trips” typically cause rendering stalls or other performance problems

OpenGL Conventions

- OpenGL has a very specific set of naming conventions
  - Each function, type, or enumerant must adhere to a set of rules defined in the spec
  - Some of these conventions make up for the fact that C does not have function overloading
  - Some of these conventions hide platform-dependent details

OpenGL Conventions: Types

- Data type names...
  - Begin with `GL`
  - Have an associated function suffix
    - More on this later
  - Have a defined bit-size
    - The bit-size is the same on *all* platforms
  - Integer types may be signed or unsigned
    - Unsigned types get a `u` after the `GL`

OpenGL Conventions: Types

GL Type Name	Common C Type	Bit-size	Notes
GLbyte	char	8-bits	
GLshort	short	16-bits	
GLint	int	32-bits	May be long
GLubyte	unsigned char	8-bits	
GLushort	unsigned short	16-bits	
GLuint	unsigned int	32-bits	May be unsigned long
GLfloat	float	32-bits	Single precision float
GLdouble	double	64-bits	Double precision float
GLboolean	unsigned char	8-bits	
GLclampf	float	32-bits	Implies range [0, 1]

- See page 14 of the OpenGL 3.0 spec for the complete list of types

OpenGL Conventions: Enumerants

- Enumerant (*enum* for short) names...
  - Begin with `GL_`
  - Are all upper-case
  - Separate words with underscores
- When passed as function parameters, enums have the type `GLenum`
- Examples:
  - `GL_VERTEX_SHADER`, `GL_ARRAY_BUFFER`, `GL_TRIANGLES`

OpenGL Conventions: Functions

- Function names...
  - Begin with `gl`
  - Begin new words with a capital letter
    - Sometimes called “camel case”
  - Remaining letters in words are lower-case
  - May have suffixes that specify the type and count of parameters

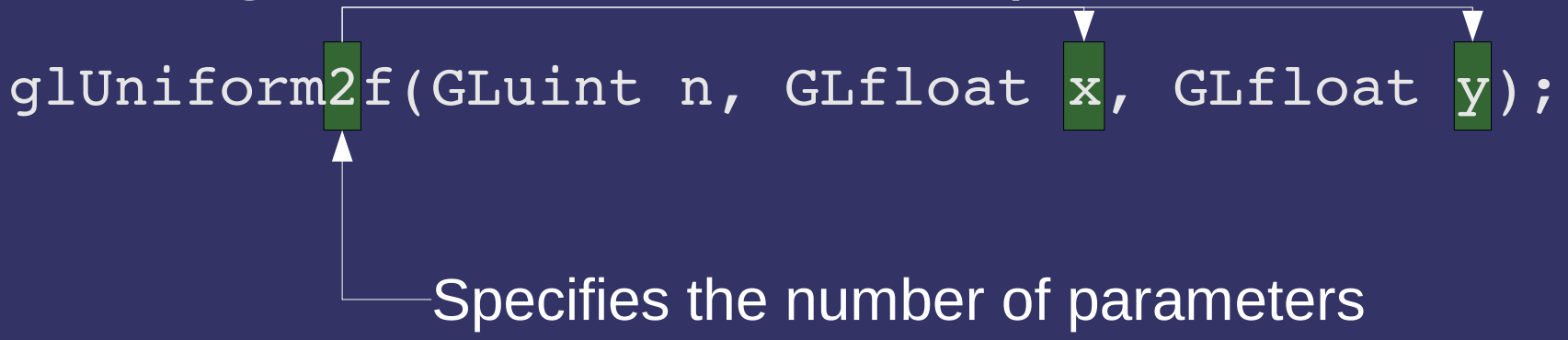
OpenGL Conventions: Functions

- Single-signature function examples:
  - `glClear`, `glDrawArrays`, `glCompileShader`
- Multi-signature function examples:

```
glUniform2f(GLuint n, GLfloat x, GLfloat y);
```

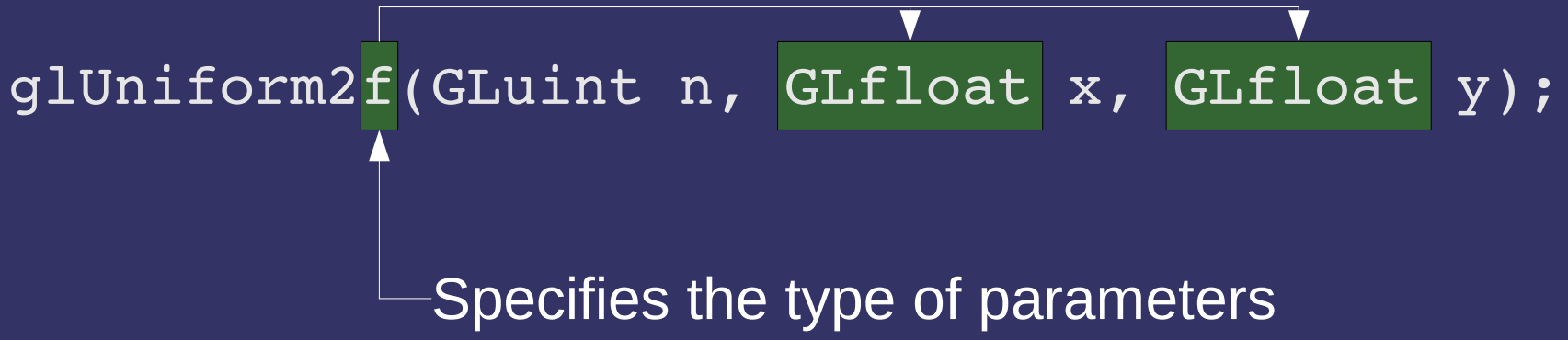
OpenGL Conventions: Functions

- Single-signature function examples:
  - `glClear`, `glDrawArrays`, `glCompileShader`
- Multi-signature function examples:



OpenGL Conventions: Functions

- Single-signature function examples:
  - `glClear`, `glDrawArrays`, `glCompileShader`
- Multi-signature function examples:



OpenGL Conventions: Functions

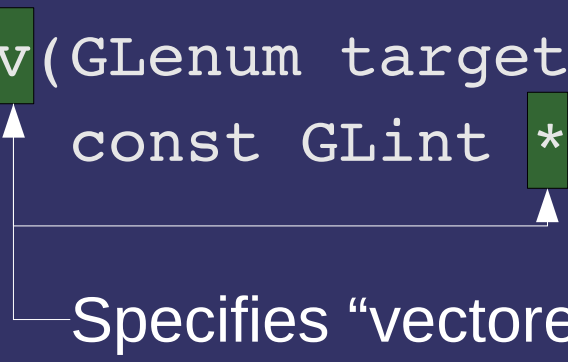
- Single-signature function examples:
  - `glClear`, `glDrawArrays`, `glCompileShader`
- Multi-signature function examples:

```
glUniform2f(GLuint n, GLfloat x, GLfloat y);
glTexParameteri(GLenum target, GLenum pname,
                GLint param);
glTexParameteriv(GLenum target, GLenum pname,
                 const GLint *param);
```

OpenGL Conventions: Functions

- Single-signature function examples:
  - `glClear`, `glDrawArrays`, `glCompileShader`
- Multi-signature function examples:

```
glUniform2f(GLuint n, GLfloat x, GLfloat y);
glTexParameteri(GLenum target, GLenum pname,
                GLint param);
glTexParameteriv(GLenum target, GLenum pname,
                 const GLint *param);
```



References

- General OpenGL and OpenGL specs:

  http://www.opengl.org/

  http://www.opengl.org/documentation/specs/

- The International Obfuscated C Code Contest:

  http://www.ioccc.org/

What OpenGL does not do

- OpenGL *only* provides access to 3D graphics hardware functionality
- Common functionality that is outside its scope:
  - Loading 3D model files
  - Loading image files
  - Processing input
  - Opening windows

Window System Interface

- OpenGL is a *low-level*, device independent, platform independent graphics hardware interface
  - Window management and user I/O fall under the purview of the underlying operating system
  - A platform-dependent *window system interface* connects window system entities with OpenGL
    - Windows has WGL, X-Windows has GLX, Mac OS X has CGL, and embedded systems have EGL
  - Cross-platform apps commonly use separate libraries to bridge these differences

SDL Introduction

> "Simple DirectMedia Layer is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer.[1]"

- What does that mean for us?
  - *Lots* of web sites have OpenGL example code that uses SDL
  - We don't have to learn how to work directly with Windows for windows or user I/O
  - I use Linux, so code that I write will be useful to you

[1] From http://www.libsdl.org/

SDL Introduction (cont.)

- SDL gives us a platform independent way to interact with platform-dependent issues
  - OpenGL makes the 3D part platform-independent, *but that's it*
  - At the very least, we need to open a window and process some keyboard input

Using SDL

- Initialize the SDL library:

```
if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER) != 0) {
    exit(1);
}
atexit(SDL_Quit);
```

Using SDL – Creating a Surface

- Tell SDL what sort of window is needed:
  - Set window size, color depth, etc.
  - Use `SDL_GL_SetAttribute`

```
/* Request at least 8-bits of red. */
SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 8);

/* Request at least 8-bits of alpha. */
SDL_GL_SetAttribute(SDL_GL_ALPHA_SIZE, 8);

/* Request a double buffered surface. */
SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
```

Using SDL – Creating a Surface

- After describing the window, open it
  - Specify a couple more attributes
  - Use `SDL_SetVideoMode`

```
/* Open a double-buffered 640x480 window.  Use
 * the default color depth (set previously).
 */
SDL_SetVideoMode(640, 480, 0,
                 (SDL_OPENGL
                  | SDL_RESIZABLE));
```

Using SDL – Creating a Surface

- After describing the window, open it
  - Specify a couple more attributes
  - Use `SDL_SetVideoMode`

```
/* Open a double-buffered 640x480 window.  Use
 * the default color depth (set previously).
 */
SDL_SetVideoMode(640, 480, 0,
                 (SDL_OPENGL
                  | SDL_RESIZABLE));
```

Enable OpenGL rendering

Allow the user to resize the window

Using SDL – Events

- SDL provides input as a series of *events*
  - `SDL_WaitEvent` blocks until an event is received
  - `SDL_PollEvent` always returns immediately
- Each event has a *type*
  - Key press events have type `SDL_KEYDOWN`
  - If no real event is available, the event type returned by `SDL_PollEvent` is `SDL_NOEVENT`
- Events may have a data payload depending on the type
  - Keycode of the pressed key, etc.

Using SDL – Events

```
SDL_PollEvent(&e);
switch (e.type) {
case SDL_KEYDOWN: {
    switch (e.key.keysym.sym) {
    case 'q':
        exit(0);
    }
    break;
}
```

Using SDL – Timers

- Set a timer to trigger a callback function

```
SDL_TimerID timer_id =
    SDL_AddTimer(10, timer_callback, data);
if (timer_id == NULL)
    /* … error path … */
```

Using SDL – Timers

- Set a timer to trigger a callback function

```
SDL_TimerID timer_id =
    SDL_AddTimer(10, timer_callback, data);
if (timer_id == NULL)
    /* … error path … */
```

This function is called every 10ms and is passed this parameter

Using SDL – Timers

- We really want a timer event
  - Generate an event from the timer callback!

```
Uint32 timer_callback(Uint32 interval, void *not_used)
{
    SDL_Event e;

    e.type = SDL_USEREVENT;
    e.user.code = 0;
    e.user.data1 = NULL;
    e.user.data2 = NULL;
    SDL_PushEvent(& e);

    return interval;
}
```

Using SDL – Timers

- To play animations, we need to know how much time has elapsed since the last frame
  - We may have rotations that are measured in “degrees per second”

```
static Uint32 t0 = ~0;
…
Uint32 ticks = SDL_GetTicks();
Uint32 delta = (t0 != (Uint32)~0) ? (ticks – t0) : 0;
float dt = float(delta) / 1000.0;

t0 = ticks;
```

References

Tutorial for SDL for OpenGL:

http://gpwiki.org/index.php/C:SDL_OGL

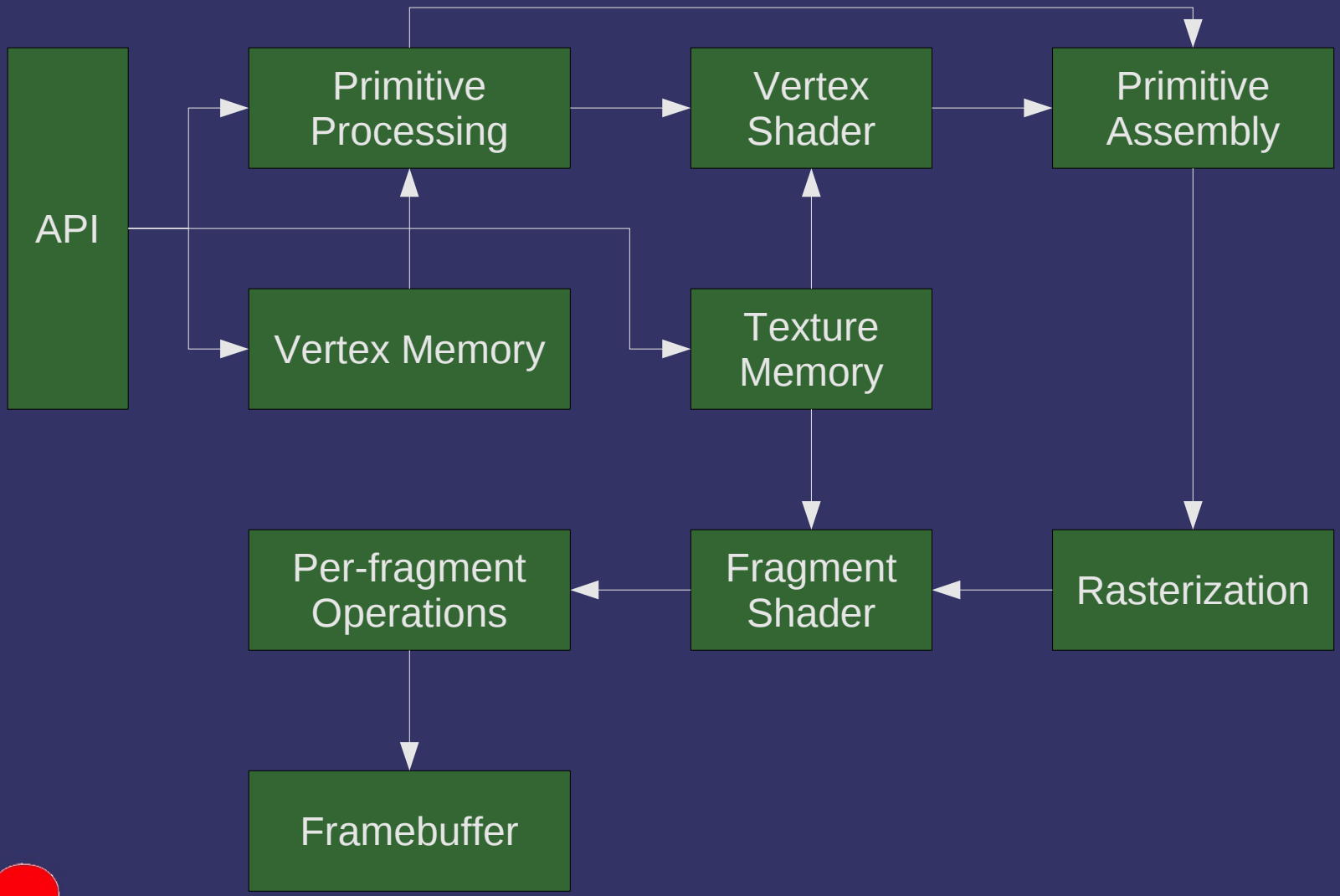
Tutorial for SDL for OpenGL on Mac OS X:

http://www.meandmark.com/sdlopenglpart1.html

Comparison of OpenGL window system interfaces:

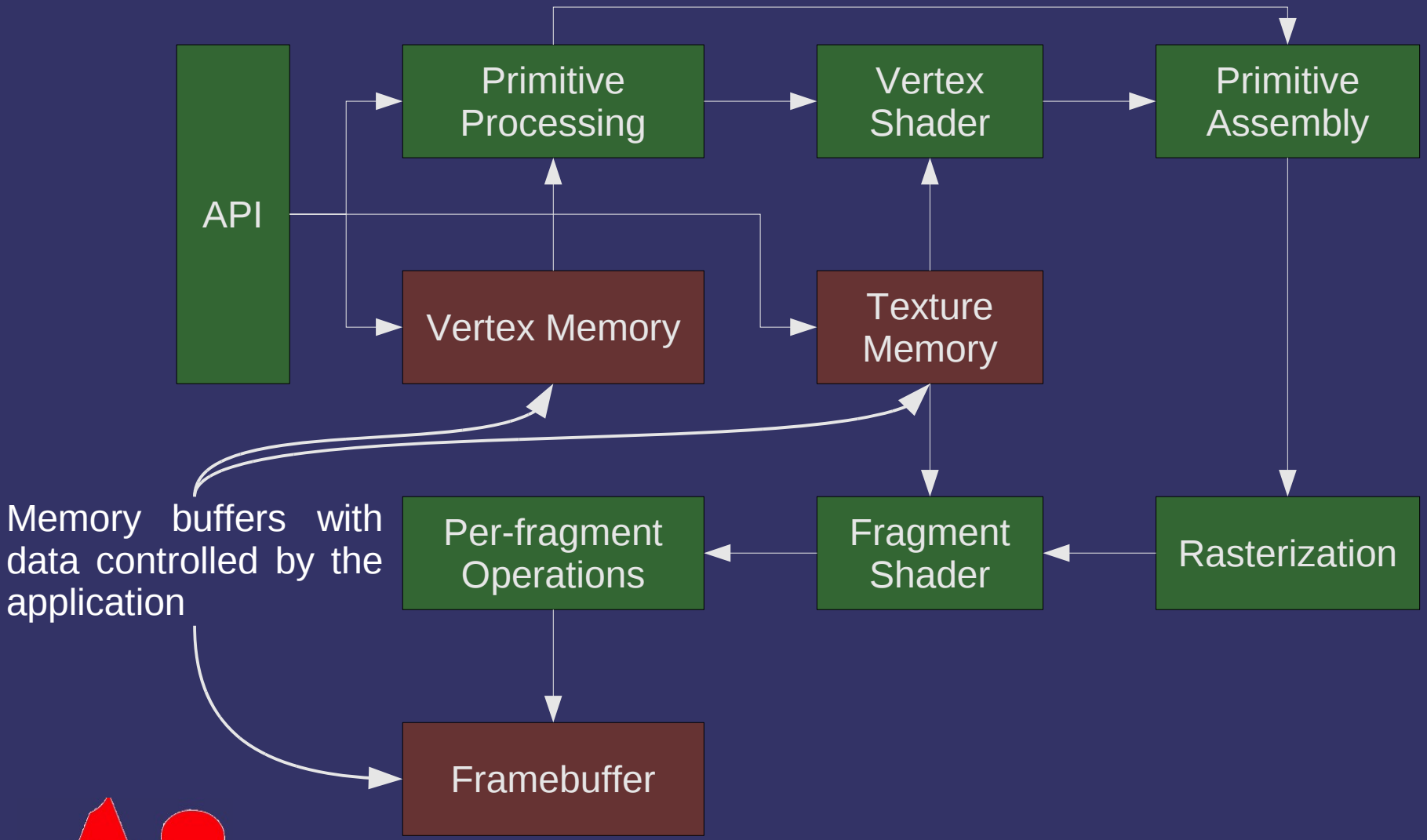
http://www.mesa3d.org/brianp/sig97/compare.htm

Graphics Pipeline



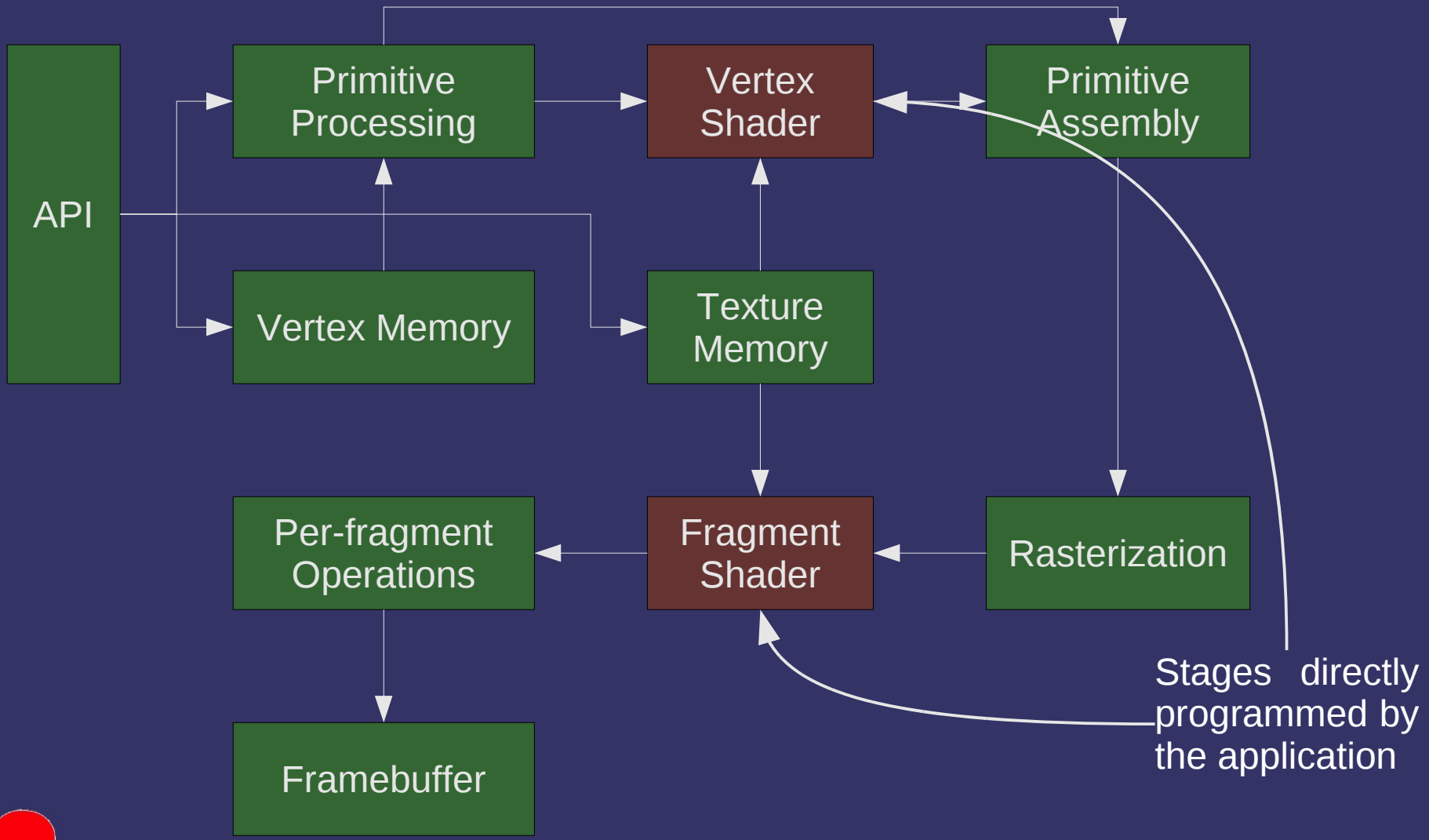
6-April-2010

Graphics Pipeline



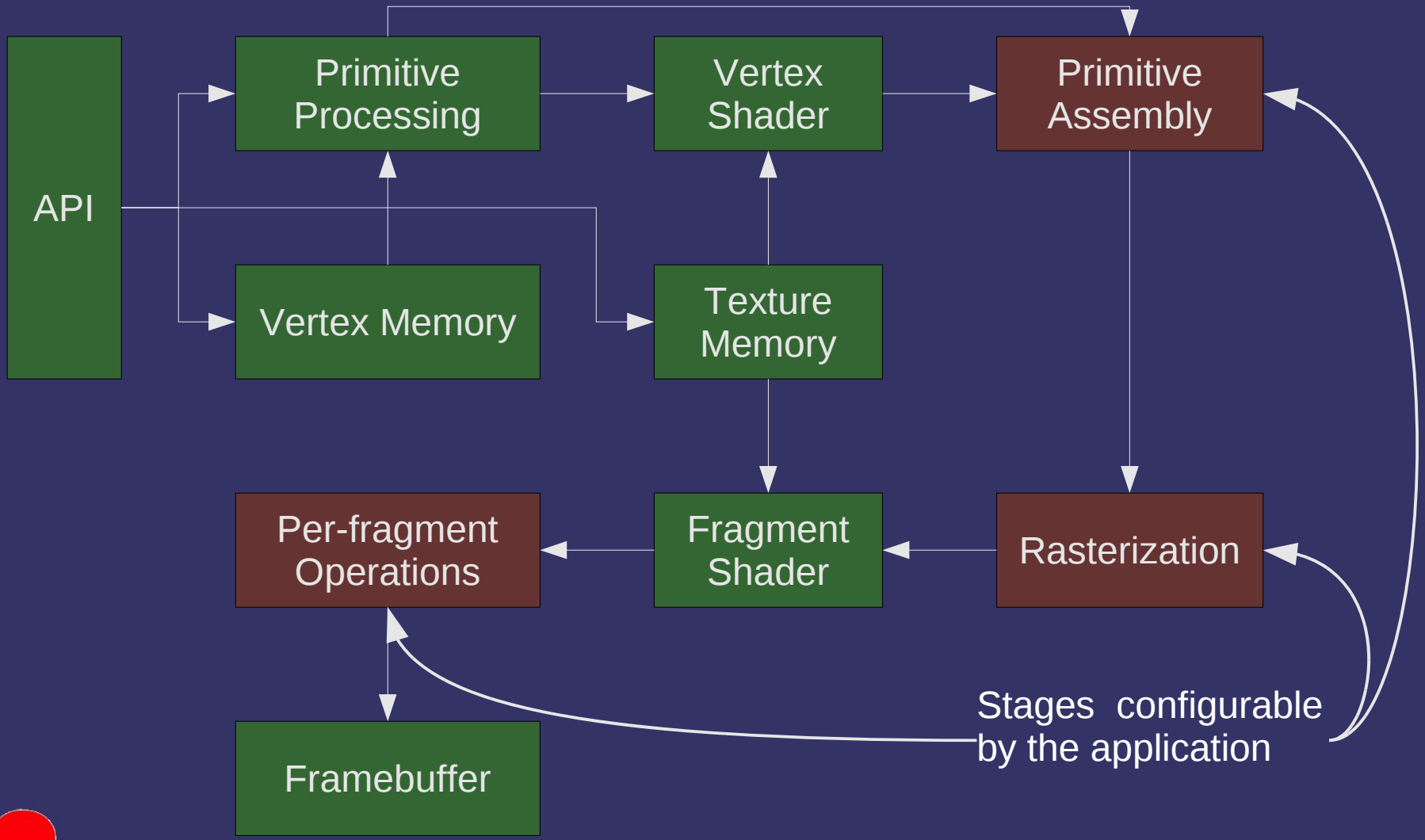
6-April-2010

Graphics Pipeline



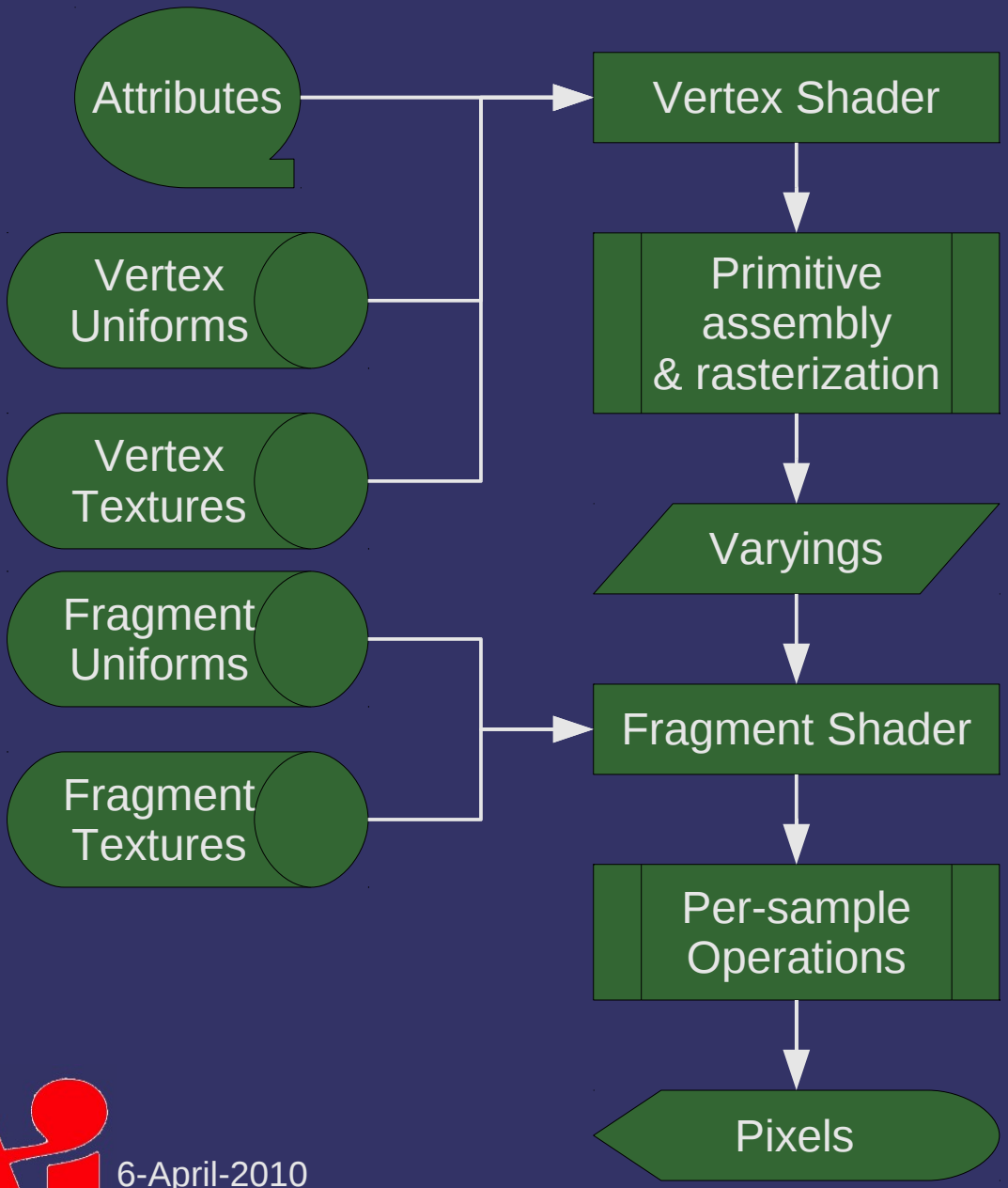
6-April-2010

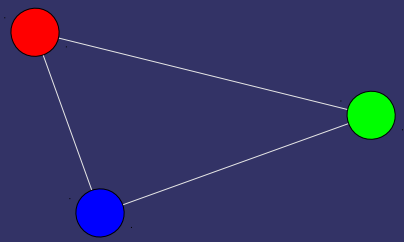
Graphics Pipeline

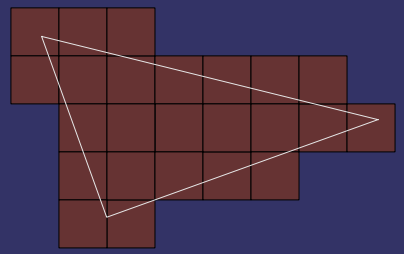


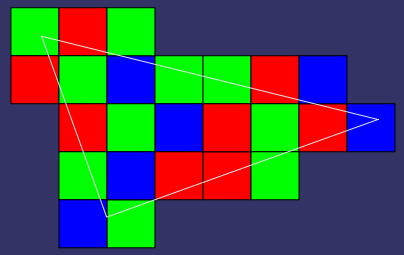
6-April-2010

Pipeline Data Flow



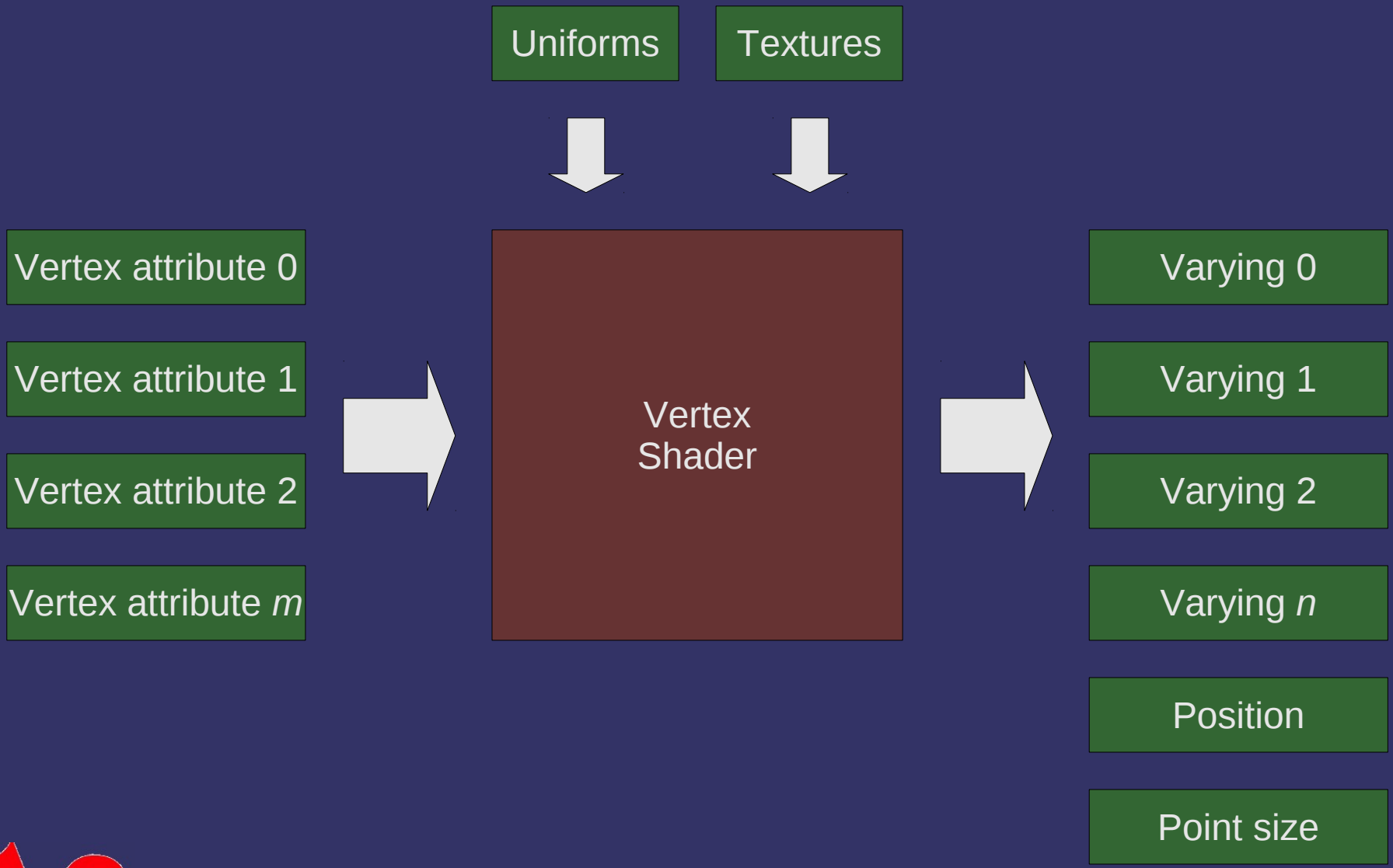






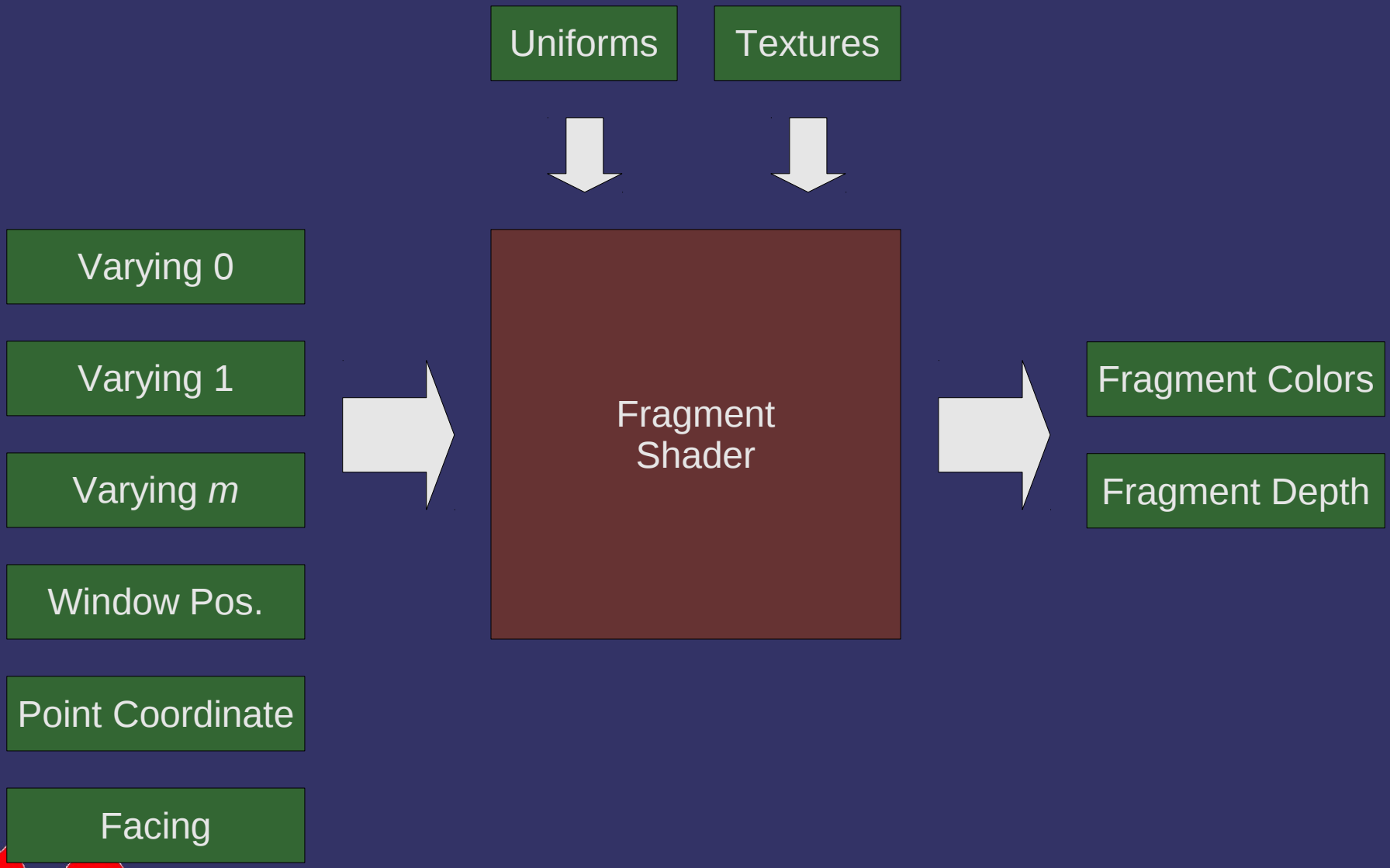
6-April-2010

Vertex Shader Environment



6-April-2010

Fragment Shader Environment



6-April-2010

GLSL – Basic Types

- 2-, 3-, and 4-element vectors of various basic types:
  - `bool` → `bvec2 bvec3 bvec4`
  - `int` → `ivec2 ivec3 ivec4`
  - `float` → `vec2 vec3 vec4`
- 2x2, 3x3, and 4x4 `float` matrices
  - `mat2 mat3 mat4`
  - Other matrix types require GLSL 1.20

GLSL – Type Qualifiers

- `uniform` – Shader inputs that are constant across a primitive group
- `attribute` – Vertex shader inputs specified per-vertex
- `varying` – Vertex outputs (fragment inputs) that are interpolated across primitives
- `const` – Local constants defined within a particular shader
  - Like `uniform`, but the value is specified in the code

GLSL – Operators

- The usual C / C++ assortment:
  - Grouping: `( )`
  - Array indexing: `[ ]`
  - Component / member selection: `.`
  - Unary: `++ — + - !`
  - Binary: `* / + -`
  - Relational: `< <= > >= == !=`
  - Selection: `?:`
  - Logical: `&& ^^ ||`
  - Sequence: `,`
  - Assignment: `= *= /= += -=`

GLSL – Flow Control

- `for`, `while`, and `do while` loops
  - Also `break` and `continue`
- `if else`
- Function calls
  - Also `return`
- `discard`
  - Terminates processing of the current fragment
  - More on this later in the term

GLSL – Functions

- Functions behave more like FORTRAN than C
  - No recursion *at all*
  - Parameters are pass-by-value, with optional copy-out
  - Extra qualifiers control parameter passing:
    - `in`: Parameter is copied in but not out. This is the default.
    - `const in`: Parameter is copied in but cannot be modified
      - May help the compiler generate better code
    - `out`: Parameter is copied out but not in
    - `inout`: Parameter is copied in and out
  - Functions can return a value
    - Or `void`

GLSL – Functions

- Pass-by value: after foo returns, what is the value of x?

```
void foo(/* in */ float a)
{
    a += 5.0;
}

void main()
{
    float x = 8.0
    foo(x);
    /* What is the value of “x” here? */
}
```

GLSL – Functions

- Pass-by value: after foo returns, what is the value of x?

```
void foo(/* in */ float a)
{
    a += 5.0;
}

void main()
{
    float x = 8.0
    foo(x);
    /* “x” is still 8.0 */
}
```

GLSL – Functions

- Pass-by value w/copy-out: after foo returns, what is the value of x?

```
void foo(out float a)
{
    a += 5.0;
}

void main()
{
    float x = 8.0
    foo(x);
    /* What is the value of “x” here? */
}
```

GLSL – Functions

- Pass-by value w/copy-out: after foo returns, what is the value of x?

```
void foo(out float a)
{
    a += 5.0;
}

void main()
{
    float x = 8.0
    foo(x);
    /* Indeterminate! */
}
```

GLSL – Functions

- Pass-by value w/copy-out: after foo returns, what is the value of x?

```
void foo(inout float a)
{
    a += 5.0;
}

void main()
{
    float x = 8.0
    foo(x);
    /* What is the value of “x” here? */
}
```

GLSL – Functions

- Pass-by value w/copy-out: after foo returns, what is the value of x?

```
void foo(inout float a)
{
    a += 5.0;
}

void main()
{
    float x = 8.0
    foo(x);
    /* “x” is 13.0 */
}
```

GLSL – Functions

- Pass-by value w/copy-out: after foo returns, what is the value of x?

```
void foo(inout float a, inout float b)
{
    a += 5.0;
    b += 10.0;
}

void main()
{
    float x = 8.0
    foo(x, x);
    /* What is the value of “x” here? */
}
```

GLSL – Functions

- Pass-by value w/copy-out: after foo returns, what is the value of x?

```
void foo(inout float a, inout float b)
{
    a += 5.0;
    b += 10.0;
}

void main()
{
    float x = 8.0
    foo(x, x);
    /* “x” is either 13.0 or 18.0 */
}
```

GLSL

And now for the stuff that is *not* like C...

6-April-2010

GLSL – Constructors

- C++-like constructor syntax for vectors, matrices, and structures:

```
vec4 color = vec4(1.0, 1.0, 1.0, 0.5);
struct foo { vec2 coord; float intensity; };
foo bar = foo(vec2(0.3, 0.6), 1.0);
```

- And arrays...

```
vec2 data[] = vec2 [] (vec2(1.0, 1.0),
                       vec2(0.5, 0.5));
```

- And *almost all* type conversions...

```
float x = calculate_something();
bool y = bool(x);
```

GLSL – Constructors

- Vector and matrix constructors just need the right number of components:

```
void foo(vec2 a, vec2 b)
{
    vec4 c(a, b);
    …
}
```

GLSL – Swizzles

- Components of a vector can have one of three component names:
  - `x`, `y`, `z`, `w` – Used for positions
  - `r`, `g`, `b`, `a` – Used for colors
  - `s`, `t`, `p`, `q` – Used for texture coordinates

GLSL – Swizzles

- Use to reorder or replicate data:

```
vec4 x;
vec2 y;

y = x.zw;
x = y.rgrg;
x = y.x;      // illegal
x = y.zw;     // illegal
y = x.sw;     // illegal
```

GLSL – Swizzles

- Use to reorder or replicate data:

```
vec4 x;
vec2 y;

y = x.zw;
x = y.rgrg;
x = y.x;      // illegal
x = y.zw;     // illegal
y = x.sw;     // illegal
```

Note: 4-components from a 2-component vector!

GLSL – Swizzles

- Use to mask and reorder writes:

```
vec4 x;
vec2 y;

y.x = x.w;
x.wz = y.rg;
y.w = x.x;   // illegal
x.xx = y;    // illegal
x.yz = y.rrr;// illegal
```

References

- GLSL quick reference

  http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf

- GLSL language spec

  http://www.opengl.org/documentation/specs/

- A couple diagrams earlier were adapted from Benj Lipchak's presentation at:

  http://people.freedesktop.org/~idr/GLSL_presentation/

Next week...

- Input data
  - Vertex buffers
  - Uniforms
- Transformations
  - Modeling
  - Viewing
  - Projection

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.

6-April-2010