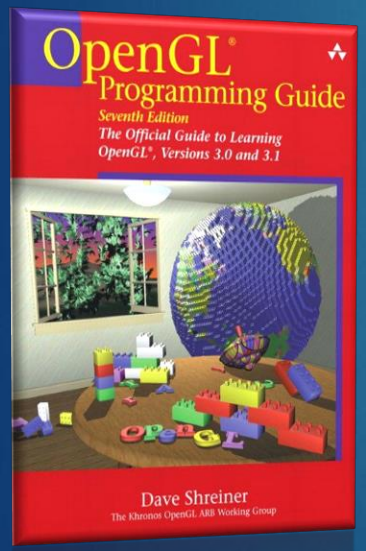
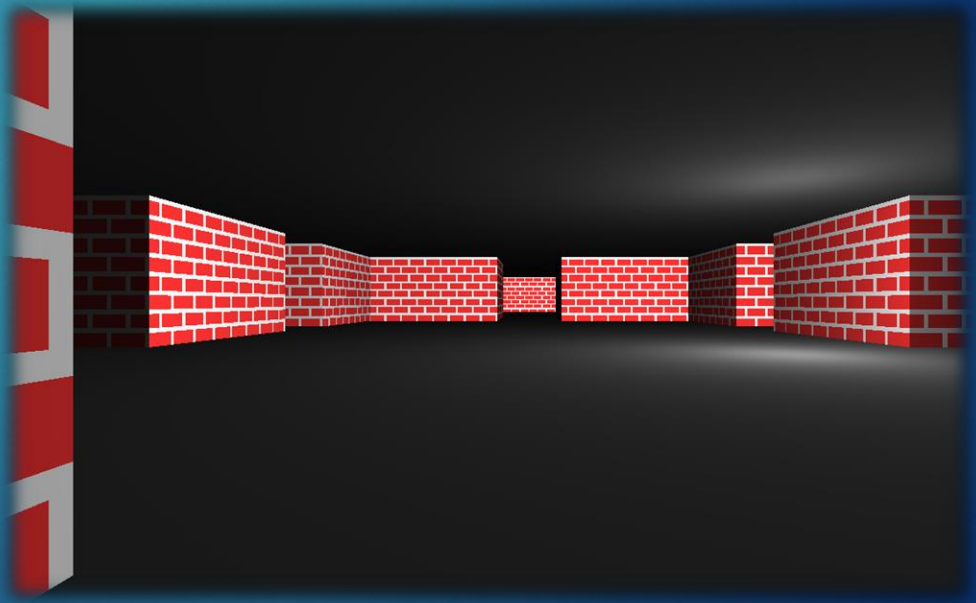
Introduction to



Sergio Ruiz

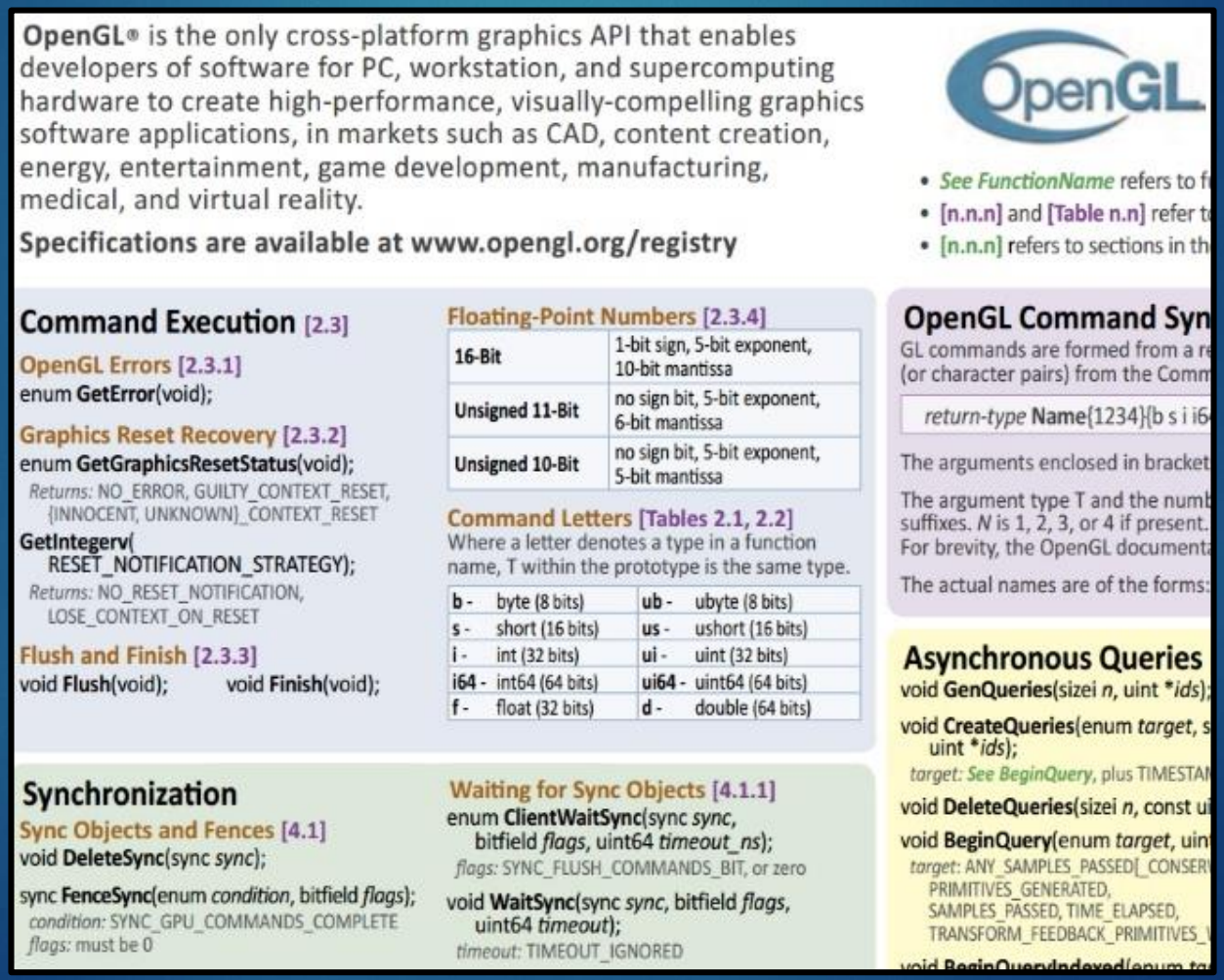
History and Design Goals

https://www.opengl.org/wiki/History_of_OpenGL



History and Design Goals

OpenGL 4.5 (2014)



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

OpenGL

- *See FunctionName* refers to f
- [n.n.n] and [Table n.n] refer t
- [n.n.n] refers to sections in th

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 Syn

GL commands are formed from a r
(or character pairs) from the Comm

return-type **Name**{1234}{b s i i6

The arguments enclosed in bracket

The argument type T and the numb
suffixes. *N* is 1, 2, 3, or 4 if present.
For brevity, the OpenGL documenta

The actual names are of the forms:

Asynchronous Queries

void **GenQueries**(sizei *n*, uint **ids*);

void **CreateQueries**(enum *target*, s
uint **ids*);

target: See BeginQuery, plus TIMESTA

void **DeleteQueries**(sizei *n*, const ui

void **BeginQuery**(enum *target*, uin

target: ANY_SAMPLES_PASSED[_CONSER
PRIMITIVES_GENERATED,
SAMPLES_PASSED, TIME_ELAPSED,
TRANSFORM_FEEDBACK_PRIMITIVES_

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

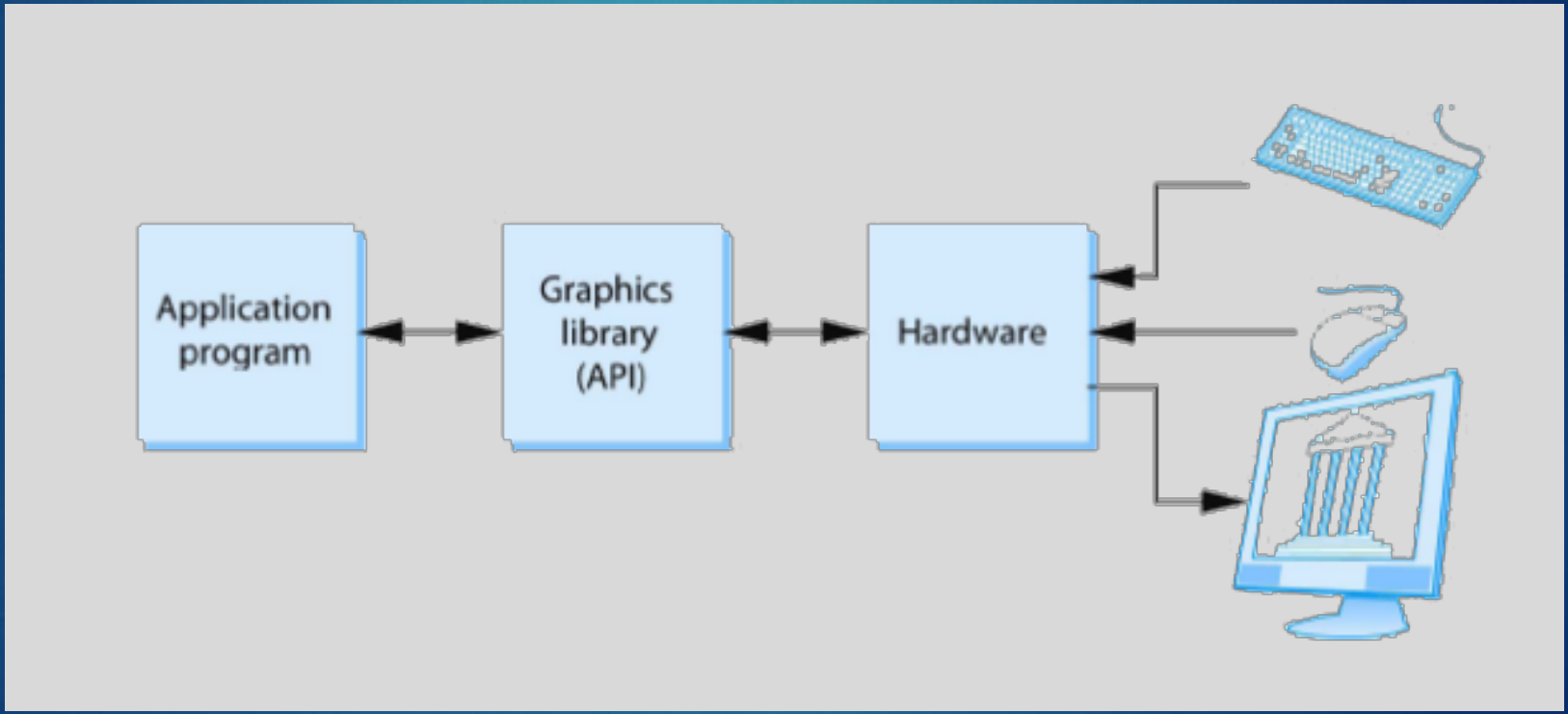
History

- In 1982 Silicon Graphics International (SGI) revolutionized graphics workstations by introducing hardware image synthesis.

- Application developers used a Graphics Library to program the system.

History

- The GL made it relatively simple to program 3D interactive applications until its success lead to the creation of OpenGL in 1992.



OpenGL Design Goals

- SGI designed OpenGL to be a graphics programming API:
  - Relatively hardware-independent.
  - Natural and compact.
  - Easy to use.
  - Close enough to the hardware for an excellent performance (using hardware acceleration).
  - Focused on rendering.
  - Independent of the window system.

OpenGL Design Goals

- To be used with C and C++ (but also compatible with Java, Python, and other programming languages through bindings).
- Free to use standard.
- Controlled and reviewed by a board including Nvidia, AMD, Intel, HP, 3DLabs, IBM and others.
- Stable.
- **Extensible (its evolution reflects new hardware capacities).**

Assignment 1 personal

1. What was the OpenGL Architecture Review Board, what is its new name and list all of its current members (25 points).
2. Explain what DirectX is, its features and differences when compared to OpenGL (30 points).
3. Enlist the main features of the 2.x, 3.x, 4.0 & 4.5 versions of OpenGL (20 points).
4. Research the main graphics processors manufacturers. Insert a table indicating the features of three recent products for each manufacturer, specifying which OpenGL version they support (25 points).

Course policies

- Submit a **PDF** file to **Blackboard**.
- Include properly formatted references (**MLA** or **APA**).
- http://bibme.org

Basic concepts

What OpenGL does

- OpenGL has become a standard because:
  - It doesn't try to do to much:
    - The library focuses on drawing images, not handling windows, modeling, sounds or high level animations.
  - It does enough:
    - Useful high performance image synthesis effects.
    - Draws Objects.
    - Specifies color models.
    - Applies Lighting.
    - Handles bitmaps and image files.
    - Maps textures.

ACM SIGGRAPH CONGRESS

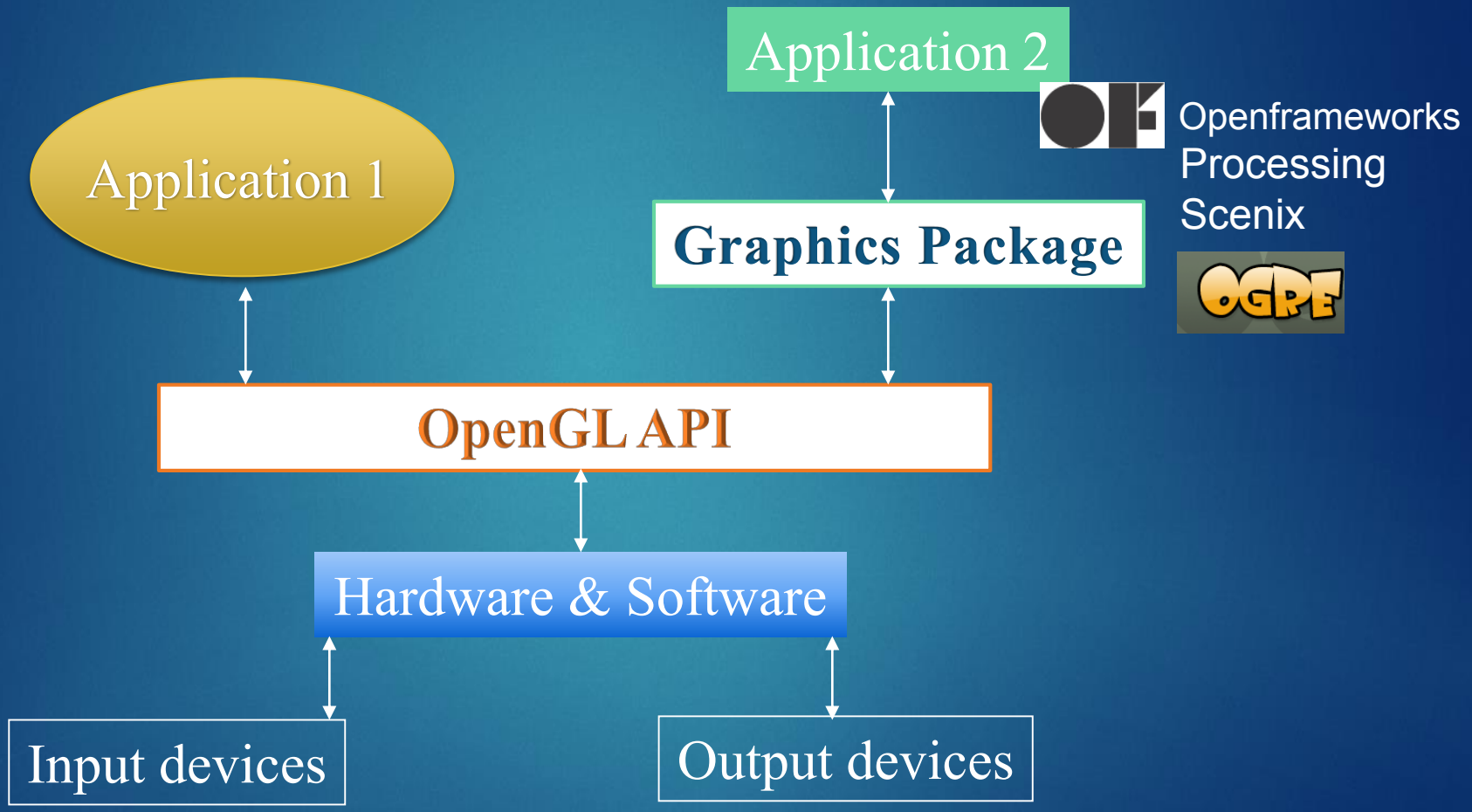
SIGGRAPH 2016
Technical Papers
trailer

https://youtu.be/dQBJ0r5Pj5s

ACMSIGGRAPH

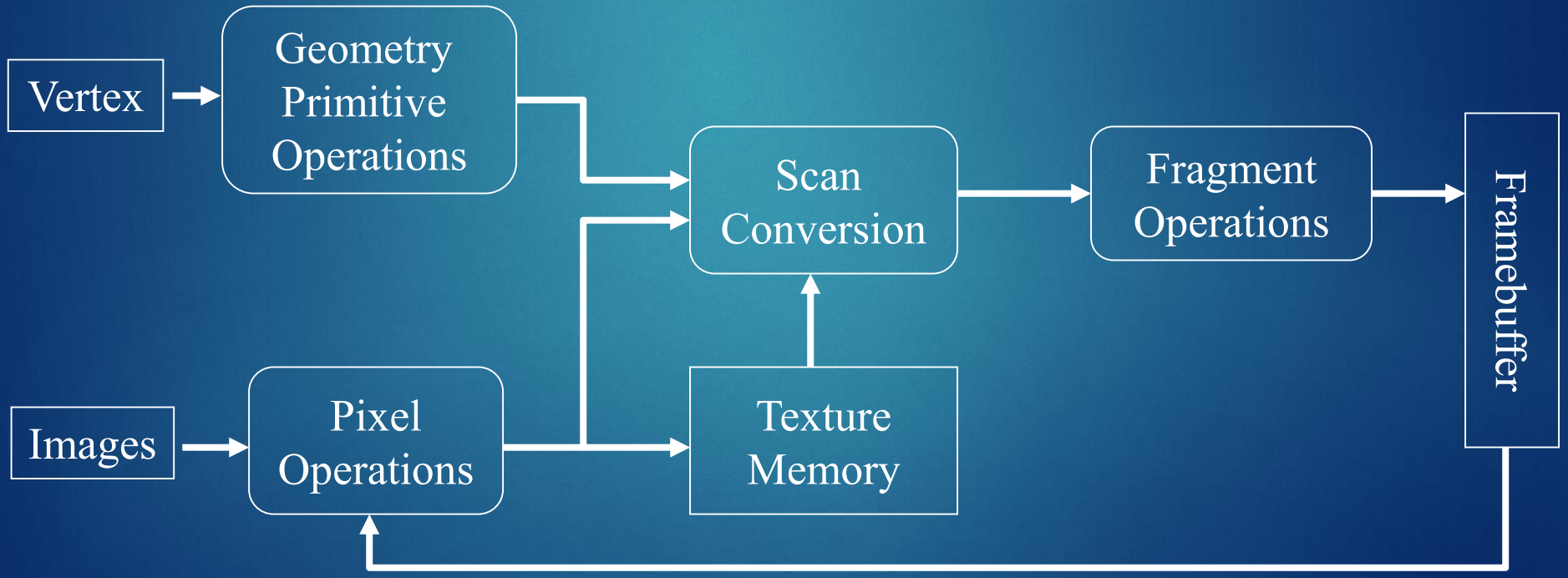
http://en.wikipedia.org/wiki/List_of_OpenGL_programs

OpenGL for programmers



OpenGL Rendering Pipeline

- Most OpenGL implementations operate following the same sequence, known as the "rendering pipeline".



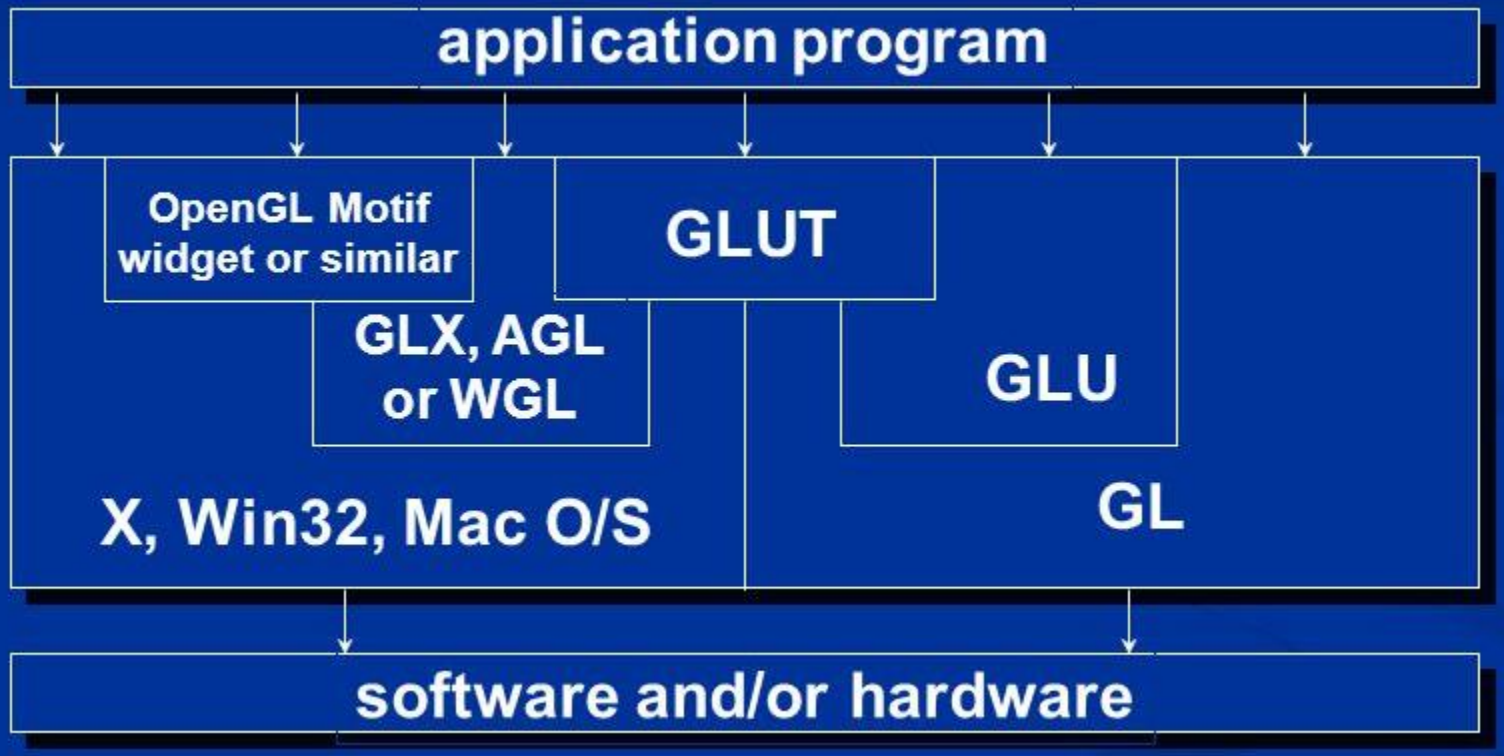
OpenGL Libraries

- OpenGL Core Library
  - OpenGL32 for Windows.
  - Provides rendering functionality.
- OpenGL Utility Library (GLU)
  - Provides higher level functionality using the Core Library.
  - Projections, surfaces, and other.

OpenGL Libraries

- OpenGL Utility Toolkit (GLUT)
  - Provides basic windowing capabilities.
  - Used for input events:
    - Mouse, keyboard, menus.
    - Callback routines.
  - Common interface for multiple platforms (Portable between X Windows, MS Windows and Mac).
  - GLUT is limited: does not include sliders, dialog boxes, menu bars, etcetera.

OpenGL Libraries



OpenGL data types

OpenGL	C	Size
`GLbyte`	`signed char`	8 bits
`GLshort`	`short`	16 bits
`GLint, GLsizei`	`long`	32 bits
`GLfloat, GLclampf`	`float`	32 bits
`GLdouble, GLclampd`	`double`	64 bits
`GLubyte, GLboolean`	`unsigned char`	8 bits no sign
`GLushort`	`unsigned short`	16 bits no sign
`GLuint, GLenum, GLbitfield`	`unsigned long`	32 bits no sign

OpenGL syntax

- Commands
  - Begin with **`gl`**
  - Camel Case:

    `glClearColor()`
- Constants
  - Begin with **`GL_`**
  - Capital letters.
  - Each word separated by **`_`**

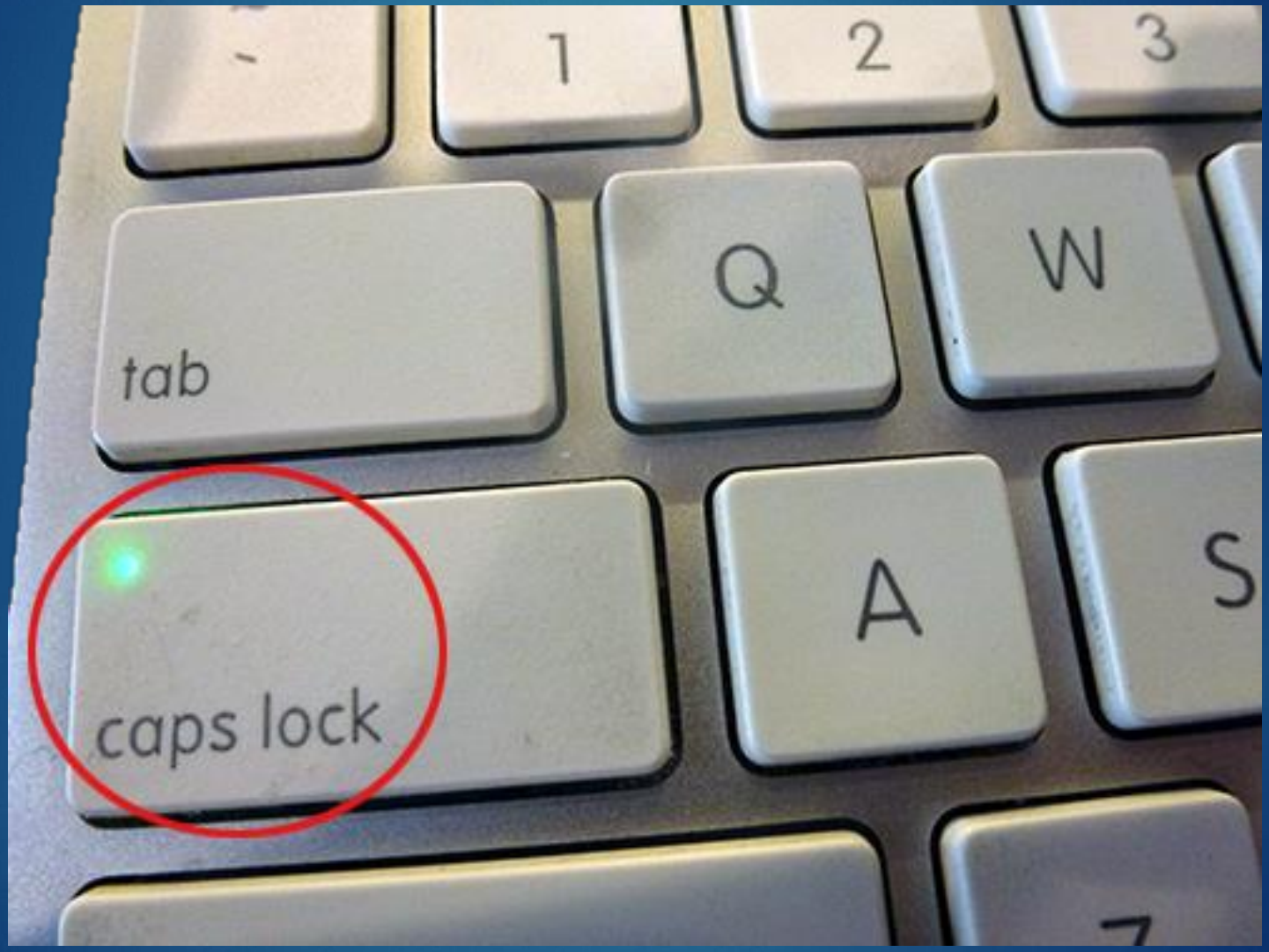
    `GL_COLOR_BUFFER_BIT`

OpenGL syntax

- Some functions use a suffix to describe arguments, like **`glColor3f()`**
  - **`3`** for the number of expected arguments.
  - **`f`** for the type of arguments. Options are:

`b`	`Glbyte`	**`ub`**	`Glubyte`
`s`	`Glshort`	**`us`**	`Glushort`
`i`	`Glint`	**`ui`**	`Gluint`
`f`	`Glfloat`	**`d`**	`GLdouble`

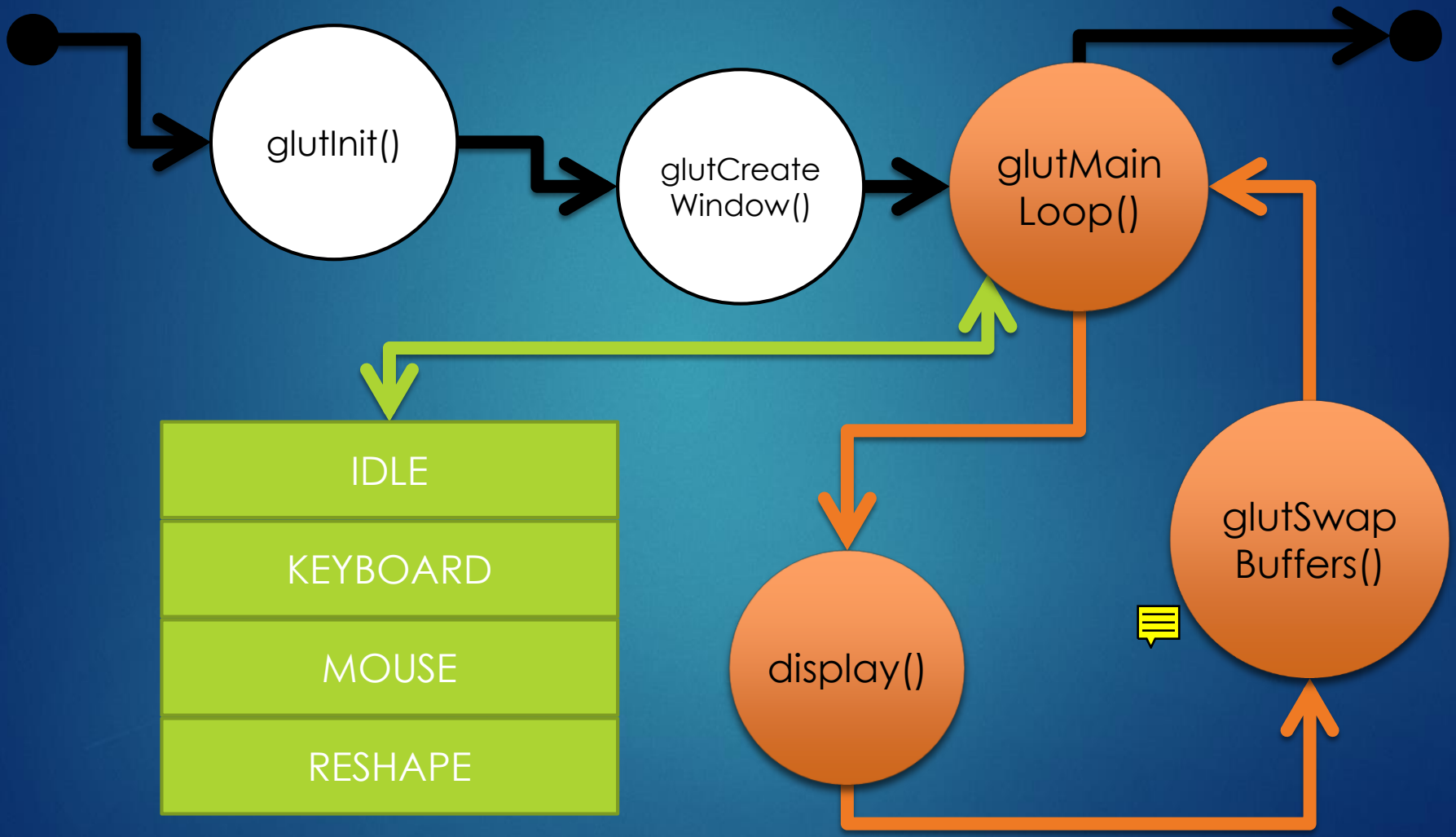
OpenGL as a state machine



OpenGL as a state machine

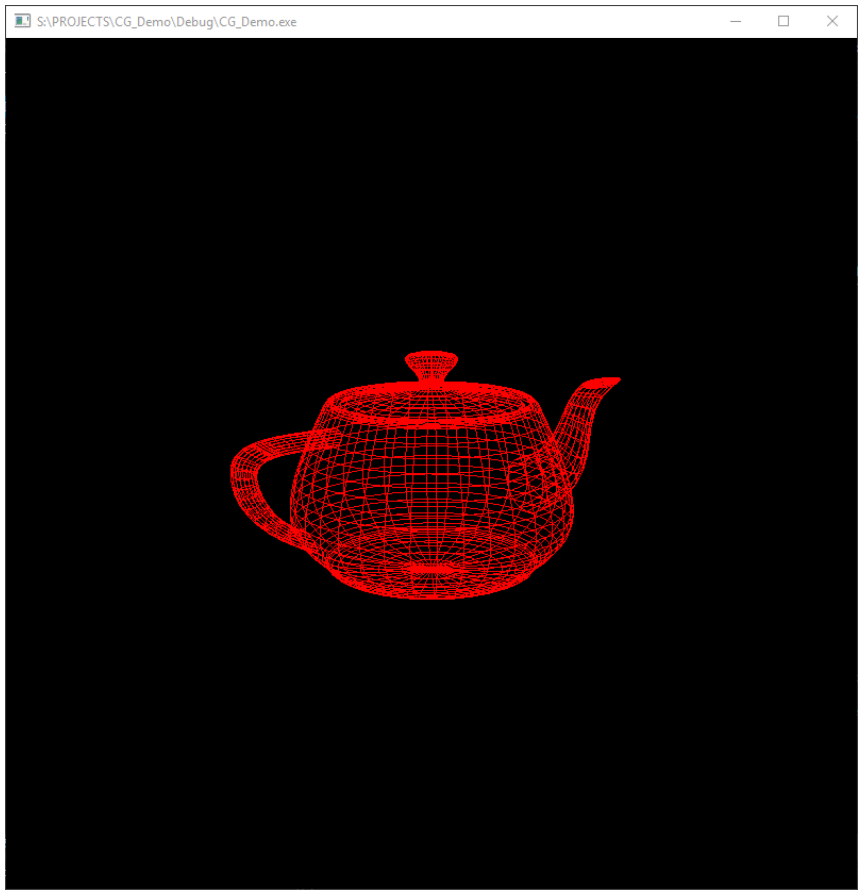
- OpenGL's states determine how objects will be drawn. States remain in effect until the programmer sets them.
- Many states are enabled or disabled using **`glEnable`** and **`glDisable`** functions.
- Rendered primitives or matrix mode are some other states that may be set.
- If you set the color state to orange, then every subsequent drawing call will also be orange.

GLUT application diagram



Example

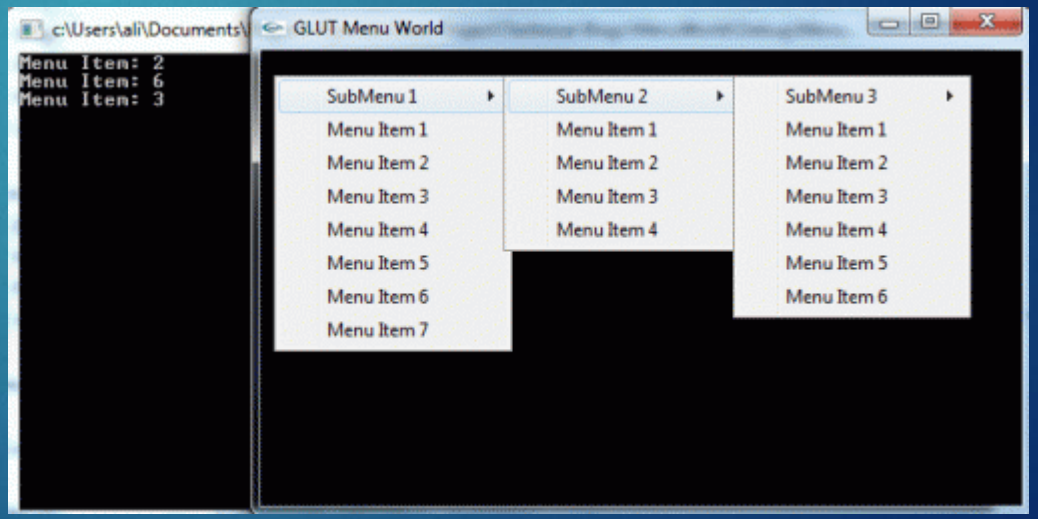
- Download and run **`CG_Demo`** from **Blackboard**.



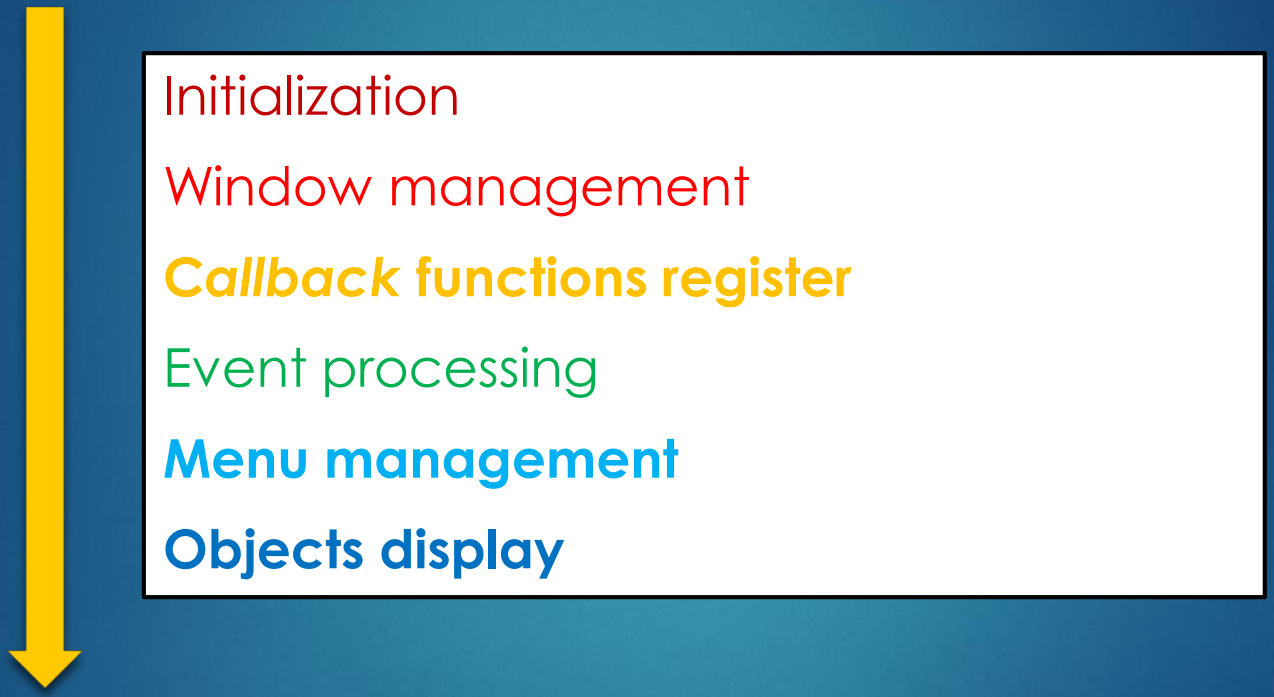
Windows and events with GLUT

What GLUT is

- OpenGL Utility Toolkit.
- Allows us to create windows for OpenGL display.
- Allows us to create **menus** to interact with the application.
- Provides several functions to display complex predefined objects.



OpenGL / GLUT application



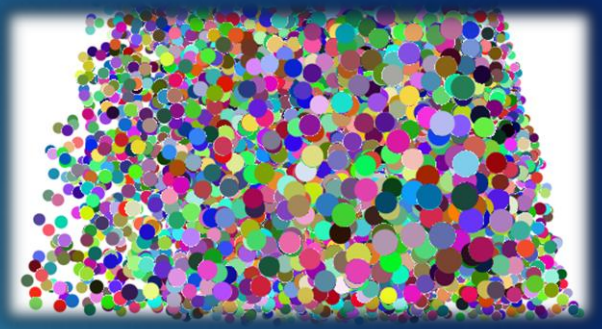
Initialization

```
glutInit(int *argcp, char **argv)
glutInitWindowPosition(int x, int y)
glutInitWindowSize(int w, int h)
glutInitDisplayMode(int mode)
```

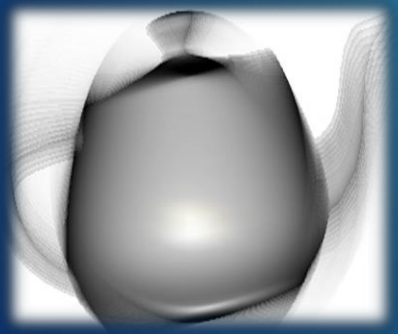
- GLUT_RGBA / GLUT_RGB / GLUT_INDEX
- GLUT_SINGLE / GLUT_DOUBLE
- GLUT_ACCUM
- GLUT_ALPHA
- GLUT_DEPTH
- GLUT_STEREO
- GLUT_STENCIL

OpenGL buffers

- **Frame / Color Buffer**: Display image.
  - Four frame/color buffers available, used for double buffering and stereographic display.
- **Alpha Buffer**: Transparency.



Depth Buffer: Distinguish close from far objects.



Accumulation Buffer: Image mix.

Stencil Buffer: Object tracking.

Window management

- **`glutCreateWindow(char *name)`**
  - Create a window titled **`name`**.
- **`glutPostRedisplay()`**
  - Redraw window contents.
- **`glutSwapBuffers()`**
  - Exchange the hidden and visible buffers.

Window management

- **`glutPositionWindow(int x, int y)`**
  - Move the window to coordinates (`x`, `y`).
- **`glutReshapeWindow(int w, int h)`**
  - Set the window size.
- **`glutFullScreen()`**
  - Show a full screen application.

Window management

- **`glutSetCursor(int cursor)`**
  - Set the pointer shape. Some options are:
    - **`GLUT_CURSOR_RIGHT_ARROW`**
    - **`GLUT_CURSOR_LEFT_ARROW`**
    - **`GLUT_CURSOR_INFO`**
    - **`GLUT_CURSOR_WAIT`**
    - **`GLUT_CURSOR_TEXT`**
    - **`GLUT_CURSOR_CROSSHAIR`**
    - **`GLUT_CURSOR_NONE`**

Callbacks

- Whenever GLUT detects an event, it calls the function assigned to respond to that event. These kind of functions are known as ***callbacks***.
- Callbacks have a set of parameters describing the event.
- Through this callback mechanism GLUT detects and responds to user and window system events.

Callback register

- **`glutDisplayFunc(void (* func)(void))`**
  - Display callback.
  - Triggered by **`glutPostRedisplay()`**
- **`glutReshapeFunc(void (*func)(int w, int h))`**
  - Triggered when the window dimensions change.
  - Used to adjust camera parameters when window size changes.

Callback register

- **`glutIdleFunc(void (* func)())`**
  - Called when display has finished and no events are detected.
  - Used to update application logic, i.e. everything not related to rendering.
- **`glutTimerFunc(unsigned int ms, void (* func)(int val), int value)`**
  - Triggered every **`ms`** milliseconds.
  - The **`value`** is sent as a parameter for **`func`**

Callback register

- **`glutKeyboardFunc(void (* func)(unsigned char key, int x, int y))`**
  - Keyboard callback, receives the pressed key and the mouse pointer position at the time.
- **`glutSpecialFunc(void (* func)(int key, int x, int y))`**
  - Called when special key events are detected:

GLUT_KEY_LEFT	GLUT_KEY_PAGE_UP	GLUT_KEY_F1
GLUT_KEY_RIGHT	GLUT_KEY_PAGE_DOWN	…
GLUT_KEY_UP	GLUT_KEY_HOME	GLUT_KEY_F12
GLUT_KEY_DOWN	GLUT_KEY_INSERT	GLUT_KEY_END

Callback register

- **`glutMouseFunc(int button, int state, int x, int y)`**
  - Called when a mouse button event is detected.
  - Buttons are defined as **`GLUT_LEFT_BUTTON`**, **`GLUT_MIDDLE_BUTTON`**, or **`GLUT_RIGHT_BUTTON`**.
  - State may be **`GLUT_UP`** or **`GLUT_DOWN`**

Callback register

- **`glutMotionFunc(void (* func)(int x, int y))`**
- **`glutPassiveMotionFunc(void (* func)(int x, int y))`**
  - Called when the mouse pointer moves.
  - **`glutMotionFunc`** is called while a mouse button is pressed, and **`glutPassiveMotionFunc`** is called otherwise.

Event processing

- `glutMainLoop()`
- Transfers program execution to the main GLUT routine (display-idle).
- Events will trigger registered callbacks.

Menu management

- **`int glutCreateMenu(void (*func)(int value))`**
  - The value passed to **`func`** should be unique for each menu element.
  - Returns the menu handle.
- **`glutSetMenu(int menu)`**
  - Sets the active menu handle.
- **`int glutGetMenu(void)`**
  - Gets the active menu handle.

Menu management

- **`glutAddMenuEntry(char *name, int value)`**
  - Adds an item to the active menu.
  - When selected, sends value to the menu callback.
- **`glutAddSubMenu(char *name, int menu)`**
  - Adds a submenu to the active menu.
- **`glutAttachMenu(int button)`**
  - Links the active menu to a mouse button, that may be **`GLUT_LEFT_BUTTON`**, **`GLUT_MIDDLE_BUTTON`** or **`GLUT_RIGHT_BUTTON`**.

Object display

- `glutSolidSphere / glutWireSphere(GLdouble radius, GLint slices, GLint stacks)`
- `glutSolidCube / glutWireCube(GLdouble size)`
- `glutSolidCone / glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks)`
- `glutSolidTorus / glutWireTorus (GLdouble innerRad, GLdouble outerRad, GLint nsides, GLint rings)`
- `glutSolidTeapot / glutWireTeapot(GLdouble size)`

Exercise

Create a window with title “OpenGL_1”.

Add a menu with the following options:

- Submenu: Solid.
  - Item: Sphere.
  - Item: Cube.
  - Item: Cone.
  - Item: Teapot.
- Submenu: Wireframe.
  - Item: Sphere.
  - Item: Cube.
  - Item: Cone.
  - Item: Teapot.
- Item: Exit.

Link the menu to the mouse Wheel click.

Exercise

The selected solid or wireframe shape will be displayed as controlled by the menu.

By left-clicking or pressing TAB the next shape option will be displayed.

Pressing the ESCAPE key will exit the application.

Drawing primitives

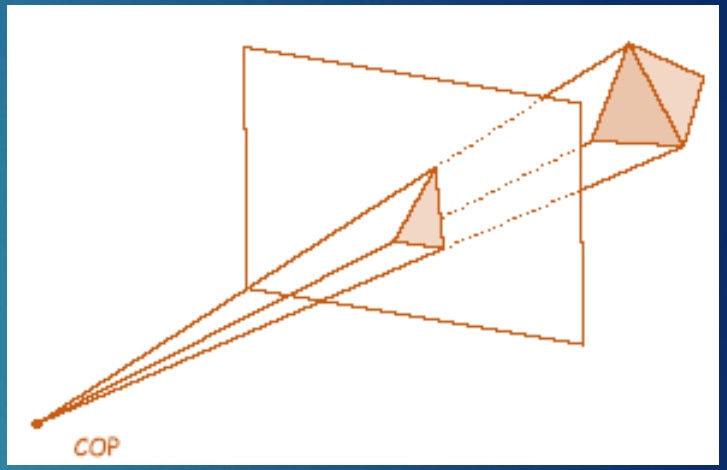
Perspective vs. Orthogonal

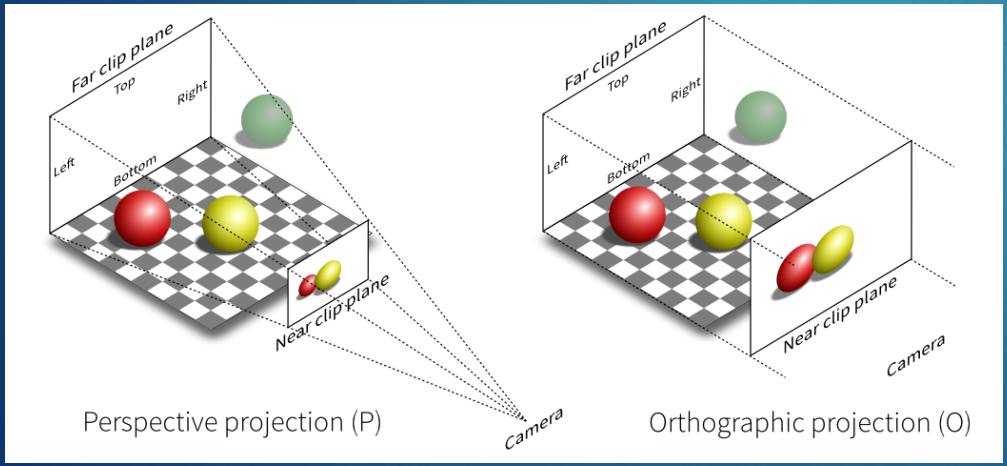
- **Perspective Projection:**

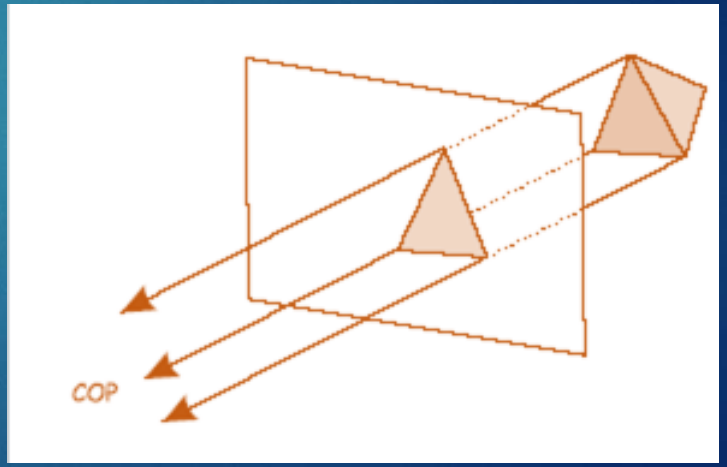
  COP is at a **finite** distance from the Projection Plane

- **Parallel Projection:**

  Distance between the Projection Plane and the COP is **infinite**



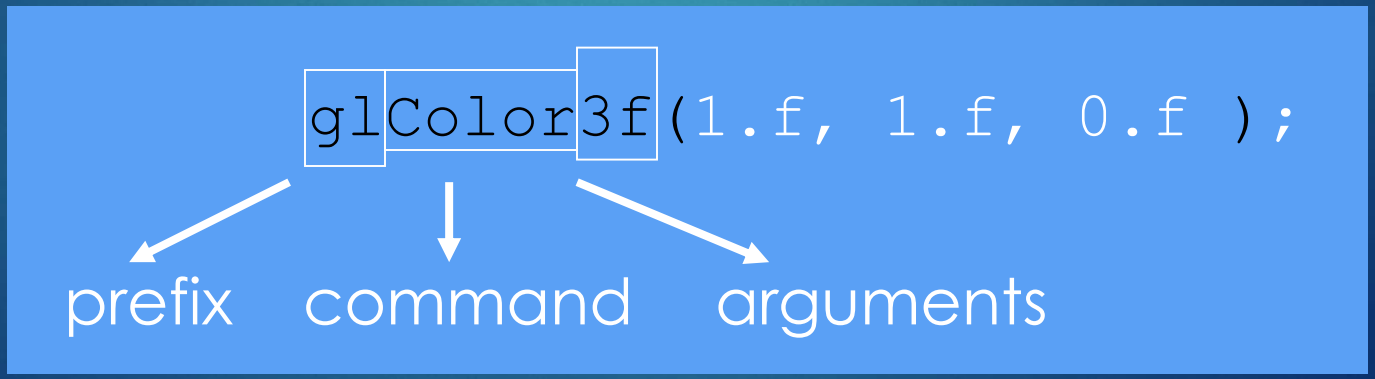




Command syntax

Type	Prefix	Example
Procedure	`gl`	`glFogf()`
Constant	`GL_`	`GL_FOG_COLOR`
Data type	`GL`	`GLfloat`

Another example:



Command syntax

Syntax "v"

Some commands end with a "v", indicating that their argument is a pointer to a vector:

```
GLfloat r=1.f, g=1.f, b=1.f;
Glfloat x[]={1,1,1};

glColor3f(r, g, b);          glColor3fv(x);
```

They both have the same final effect, what would be the performance difference?

Clearing the screen

- Before drawing primitives, you want to clear the previous frame.
- **`glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a)`**
  - Parameters indicate the clearing framebuffer color.
- **`glClear(GLbitfield mask)`**
  - Clears the specified buffers. **Mask** is a logical OR (use a single bar | to separate options) from:

```
GL_COLOR_BUFFER_BIT      GL_STENCIL_BUFFER_BIT
GL_DEPTH_BUFFER_BIT      GL_ACCUM_BUFFER_BIT
```

OpenGL primitives

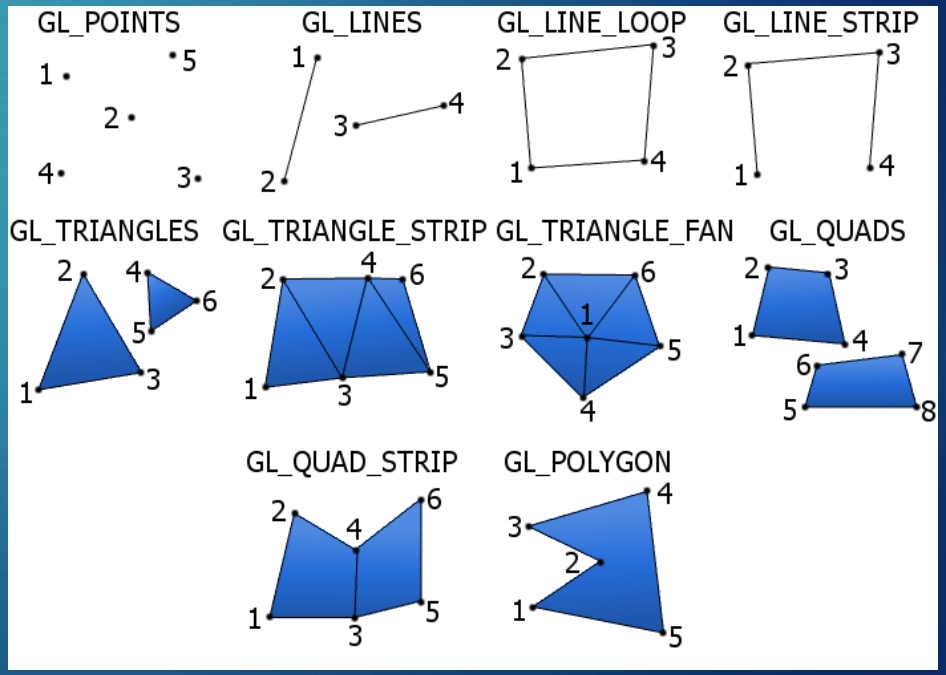
- OpenGL draws primitives according to a set of modes.
- Each primitive is defined by a group of one or more vertices.
- Each vertex may have two, three or four coordinates.
- Additional states control color, normal, material or texture coordinates for each vertex.

Begin / end paradigm

- Most OpenGL primitives are drawn as a set of vertices inside a pair of **`glBegin`** / **`glEnd`** function calls.
- The number of vertices drawn is only limited by available runtime memory.
- The type of primitives to be drawn is a parameter of **`glBegin`**.

OpenGL primitives

- **GL_POINTS**
- **GL_LINES**: separated lines.
- **GL_LINE_STRIP**: continuous lines.
- **GL_LINE_LOOP**: cyclic lines.
- **GL_POLYGON**: polygons.
- **GL_TRIANGLES**
- **GL_TRIANGLE_STRIP**
- **GL_TRIANGLE_FAN**
- **GL_QUADS**
- **GL_QUAD_STRIP**



Vertices

- Specify each vertex with the function: **`glVertex#t`**.
  - "***#***" is the number of vertex components (2, 3, 4).
  - "***t***" specifies the components types (**`s`**, **`i`**, **`d`**, **`f`**).
  - The syntax: **`glVertex#tv`** may be used to pass the components as an array.
  - Example: **`glVertex2f( 0.2f, 0.3f );`**

Vertices

- When the W component is not specified, its default is 1.0.
- When the Z coordinate is not specified, its default is 0.0.
- All internal operations in OpenGL are performed in 3D.

Example:

```
GLfloat v[]={ 30.0f, 20.0f, 0f, 1.0f };
glVertex4fv( v );
```

and

```
glVertex2f( 30, 20 );
```

are the same, why?

Primitives

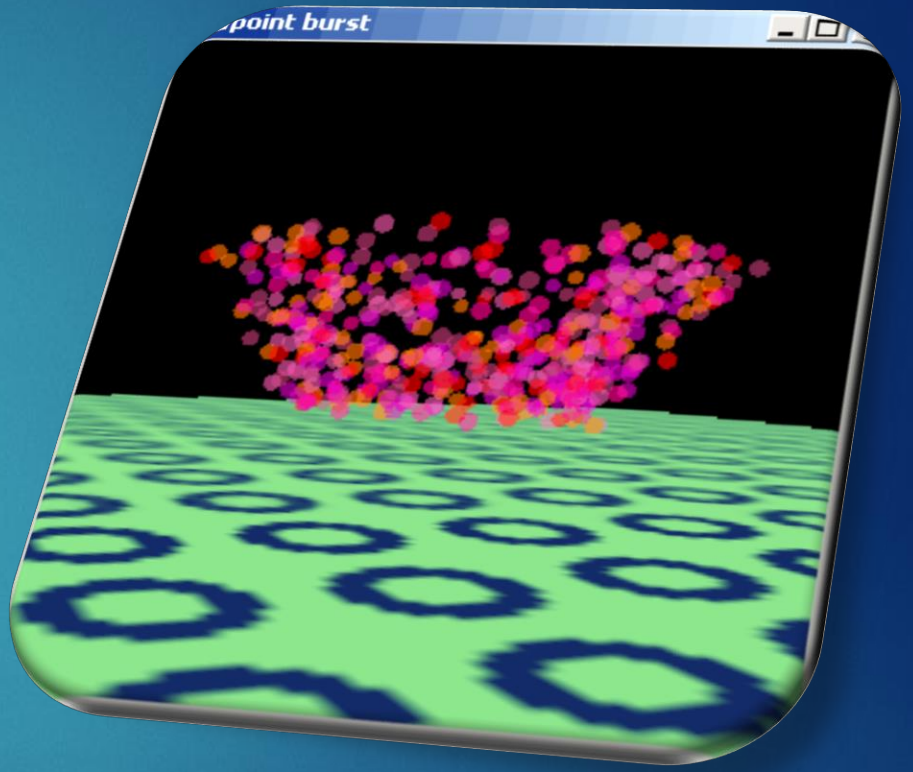
```
glBegin(GL_POINTS); // begin drawing points
  glVertex3f(0.0f, 0.0f, 0.0f);
  glVertex3f(1.0f, 0.0f, 0.0f);
  glVertex3f(1.0f, 1.0f, 0.0f);
glEnd(); // finished drawing points
```

Primitives

Can anything interesting be done with points?

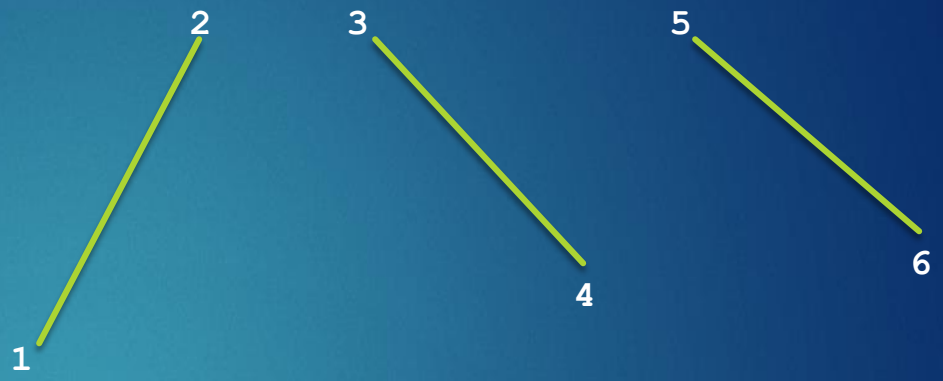
Of course!

An explosion for example.

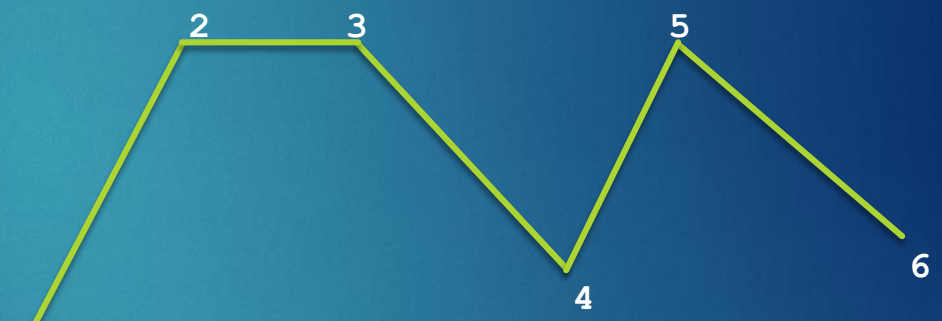


Primitives

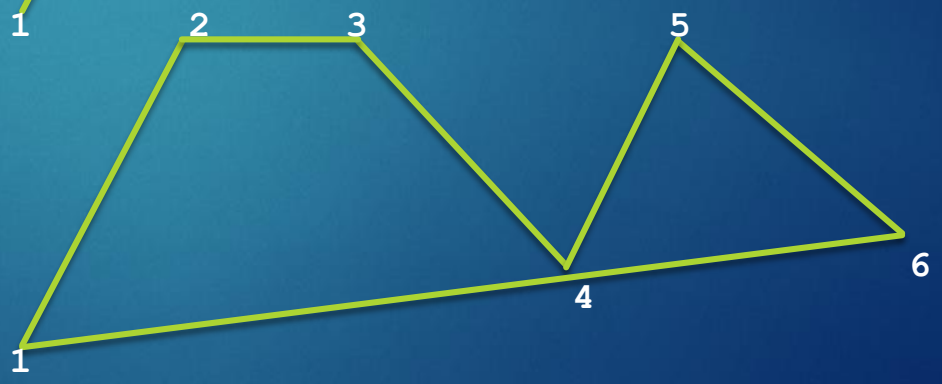
```
glBegin(GL_LINES)
```




```
glBegin(GL_LINE_STRIP)
```

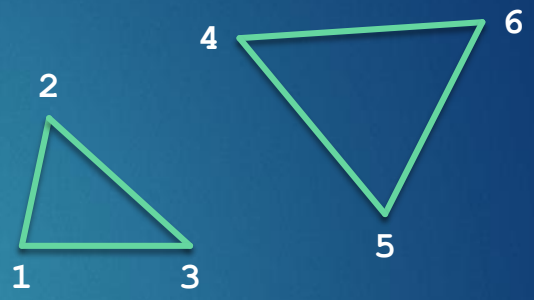



```
glBegin(GL_LINE_LOOP)
```

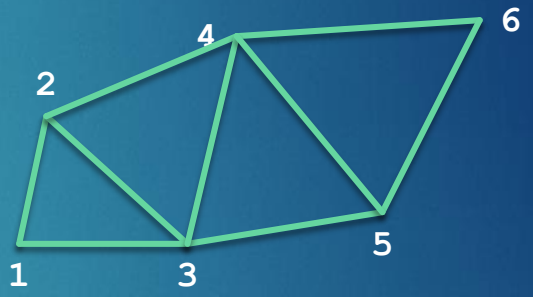


Primitives

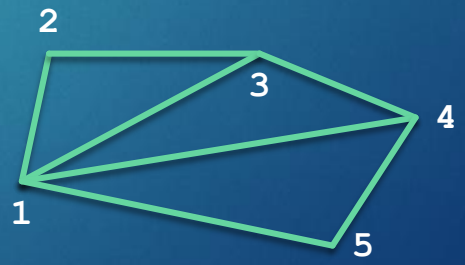
`glBegin(GL_TRIANGLES)`



`glBegin(GL_TRIANGLE_STRIP)`

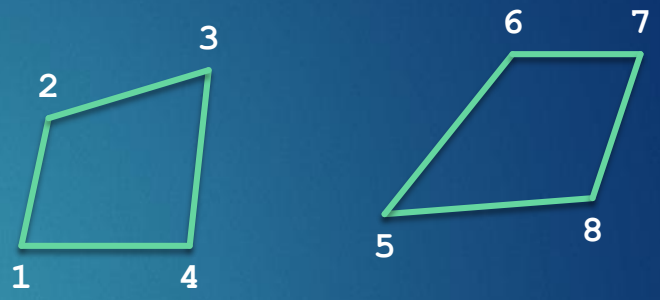


`glBegin(GL_TRIANGLE_FAN)`

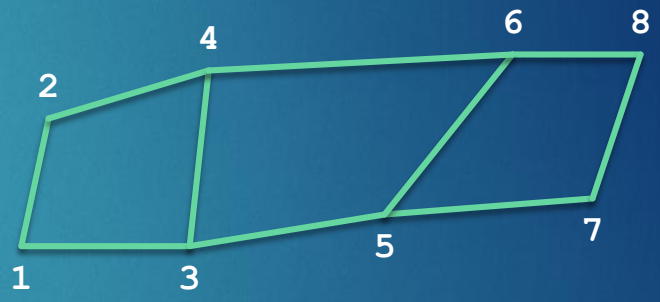


Primitives

```
glBegin(GL_QUADS)
```




```
glBegin(GL_QUAD_STRIP)
```



State modifiers

- **`glEnable(GL_DEPTH_TEST)`**
  - Distinguish between close and far objects
- **`glPointSize(GLfloat s)`**
- **`glLineWidth(GLfloat w)`**
- **`glPolygonMode(GLenum face, GLenum mode)`**
  - face: **`GL_FRONT`**, **`GL_BACK`**, **`GL_FRONT_AND_BACK`**
  - mode: **`GL_POINT`**, **`GL_LINE`**, **`GL_FILL`**
- **`glCullFace(GLenum face)`**
  - Skip drawing a face to improve performance.
  - Modified using **`glEnable(GL_CULL_FACE)`**

State modifiers

- **`GL_FRONT` / `GL_BACK`**
  - Front: vertices are distributed counter clockwise.
  - This behavior can be set using **`glFrontFace(GL_CCW | GL_CW)`**
  - For multiple primitives, the first triangle indicated is used.

State modifiers

- **`glLineStipple(GLint n, GLushort pattern)`**
  - Indicates a dotted line drawing.
  - Repeats **`pattern`** **`n`** times.
  - You need to enable **`GL_LINE_STIPPLE`**
- **`glPolygonStipple(const GLubyte *mask)`**
  - Indicates a dotted polygon drawing.
  - Mask is 32x32 bits (128 bytes)
  - You need to enable **`GL_POLYGON_STIPPLE`**

Color

- Use **`glColor#t`**.
- This function receives 3 or 4 parameters: red, green, blue and alpha.
- ***`t`*** may be of type **`b`**, **`s`**, **`i`**, **`f`**, **`d`**, **`ub`**, **`us`** or **`ui`**.
- Actual values vary depending on the data type specified.

Example

```
void Display(void)
{
  glShadeModel(GL_SMOOTH);  // color interpolation
  glClearColor(1,1,1,1);    // background color
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glBegin(GL_TRIANGLES);    // begin drawing triangles
     glColor3ub(0,0,255);   // blue state
     glVertex3f(0,0,0);     // blue vertex
     glColor3ub(255,0,0);   // red state
     glVertex3f(1,0,0);     // red vertex
     glVertex3f(0,1,0);     // red vertex
  glEnd();                  // end drawing triangles
  glFlush();  // flush fragments to the framebuffer
}
```

Shading model

- Select a shading model using

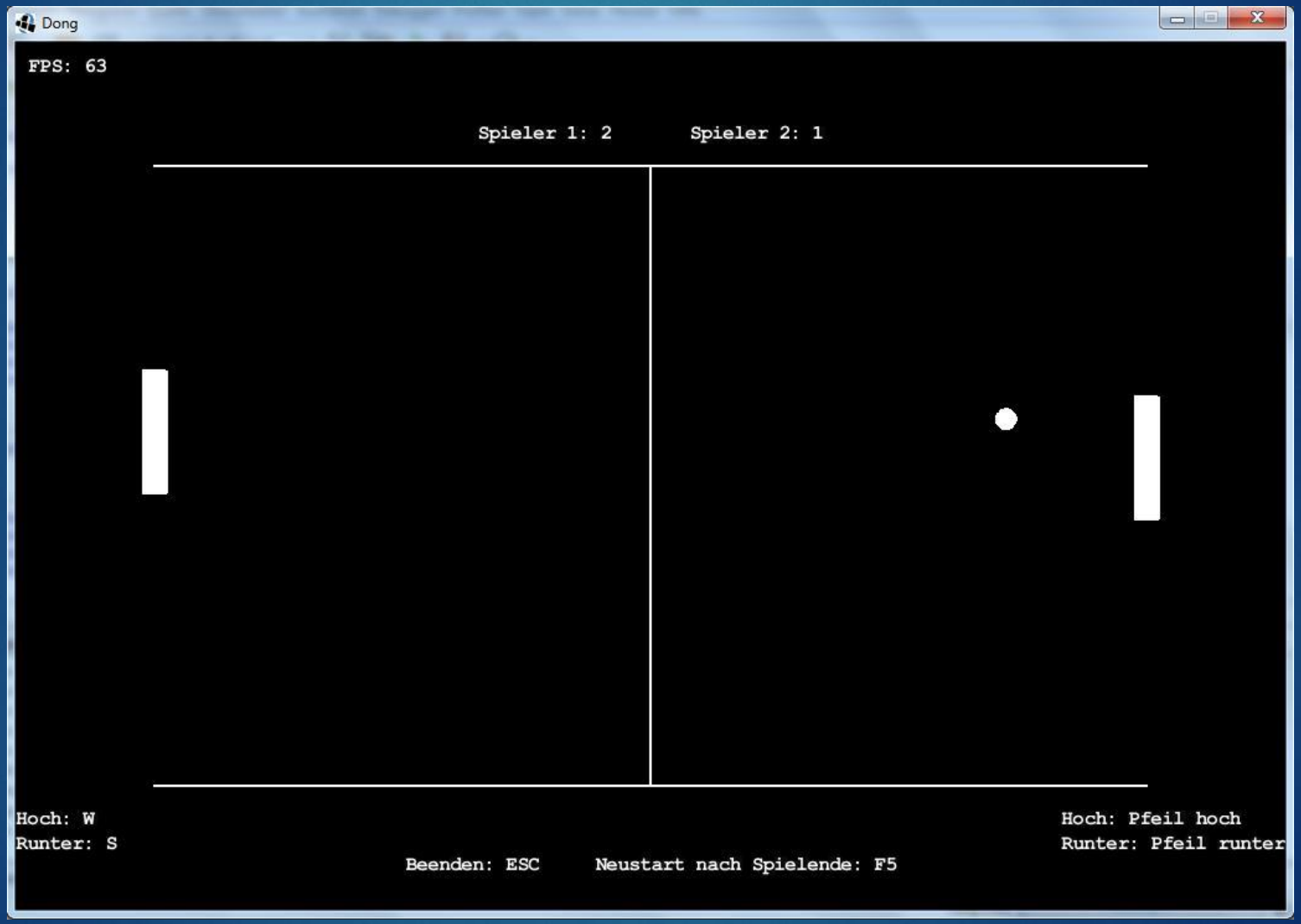
```
glShadeModel(GLenum mode)
```

  - **GL_FLAT** use a color for each polygon.
  - **GL_SMOOTH** interpolate colors between vertices of the polygon.

Exercise 2

- Pong classic arcade game.
- 1 player vs. AI.
- 1 blue and 1 red paddle. The ball is a white sphere.

Exercise 2


Dong
FPS: 63
Spieler 1: 2
Spieler 2: 1
Hoch: W
Runter: S
Hoch: Pfeil hoch
Runter: Pfeil runter
Beenden: ESC
Neustart nach Spielende: F5

Exercise 2

- When the game begins the ball moves toward a random player.
- The ball bounces off the walls.
- The paddles move up and down only.