COMP3421/9415
Computer Graphics

Introduction

Robert Clifton-Everest

Email: robertce@cse.unsw.edu.au

Course Admin

- http://www.cse.unsw.edu.au/~cs3421
- Same website for COMP9415
- See the course outline
- Using webcms for the course content, Piazza for a forum.
- Consultations Friday at 1pm in K17 G01

Lectures

- Lecture videos are linked to from the course website
- There is NO lecture in week 10
- There IS a lecture in week 13
- Guest lecture in week 6
  - Xi Ma Chen — Rendering engineer that worked on Call of Duty: WWII
- Lecture starter code is released before each lecture
  - Code along if you want

Lab

- Optional lab this week (not marked)
- Attend any session you like
- Opportunity to get your laptop setup for the practical components of the course
- Times:
  - Monday 3-4pm or 4-5pm in K14 labs (organ, piano, clavier)
  - Wednesday 3-4pm or 4-5pm in clavier
  - Will run another one if there is demand (subject to lab availability)

Tutorials

- Tutorials start week 2
  - Reenforce what we cover in the Lectures
  - Assignment partners are selected from your tutorial groups, so get to know people!
  - NO Tutorial in week 10

Assignments

- Assignment 1
  - Individual
  - 2D graphics
  - Due at the end of week 5
- Assignment 2
  - Pairs
  - 2D graphics
  - Milestone 1 due at end of week 10
  - Final milestone due at the end of week 12
  - Demonstrate in week 13

Quizzes

- 5 online quizzes throughout the course
- Released in weeks 1,3,5,7 and 9
- Due at the end of weeks 2,4,6,8, and 11
- The quiz in week 9 will be a mega-quiz. You have longer to complete it.

Assumed knowledge

- Java
  - Don’t be afraid to ask questions
- Basic linear algebra
  - Vectors, matrices
  - We will revise this

Gained knowledge

- Computer graphics (obviously)
- We will also touch on many other areas
  - Linear algebra
  - Geometry
  - High-performance computing
  - Parallelism
  - General programming

Why Graphics?

- Games

- Animation

- Special effects

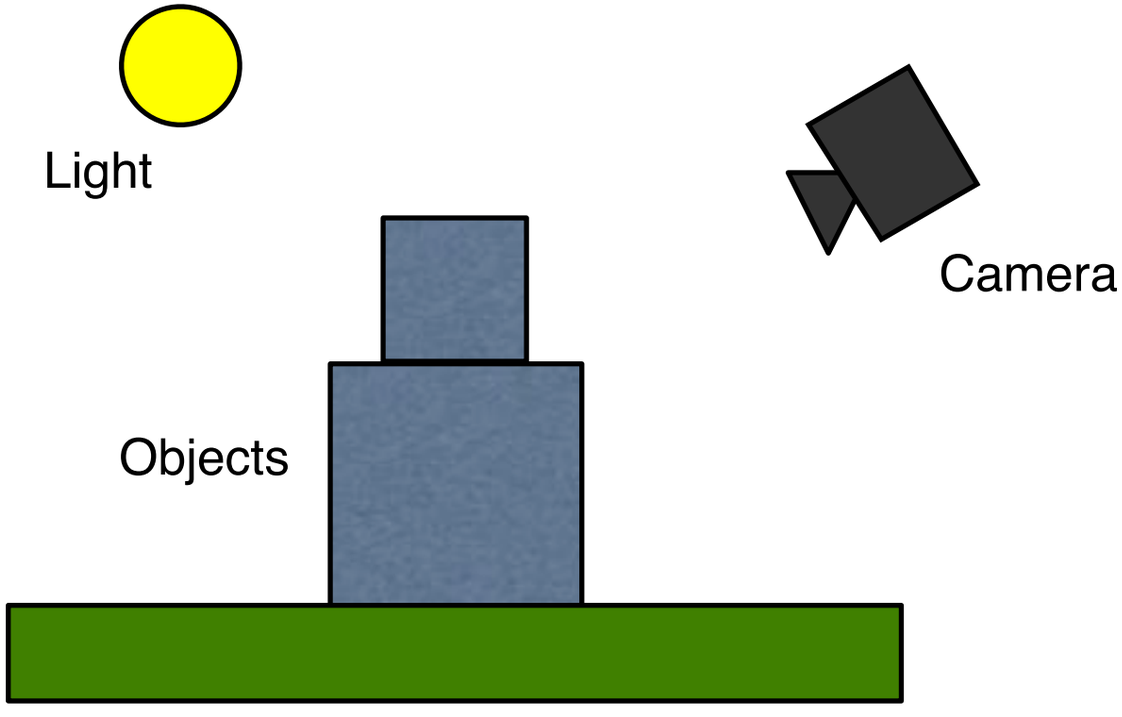
- More generally?

Graphics Then and Now

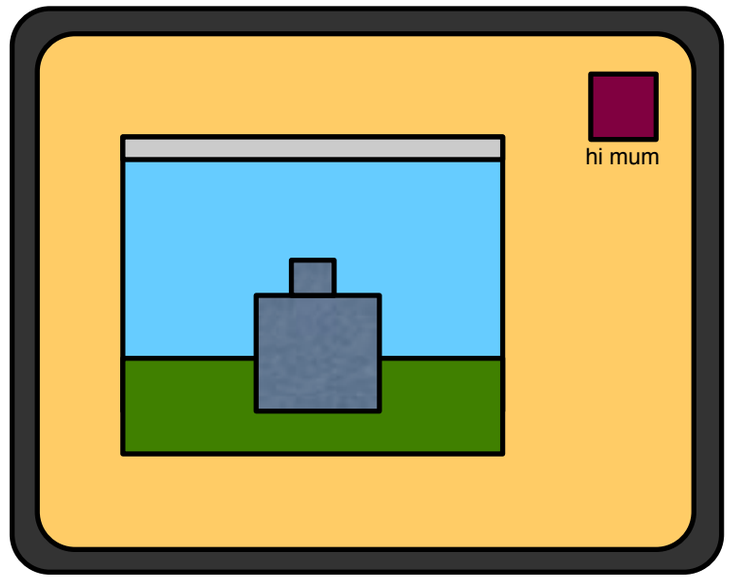
- 1963 Sketchpad (4mins 20)
  - https://www.youtube.com/watch?v=USyoT_Ha_bA
- 2017 Pixar's Renderman
  - https://www.youtube.com/watch?v=wO5hlSgYXvM

What is Computer Graphics?

- Algorithms to automatically render images from models.



model



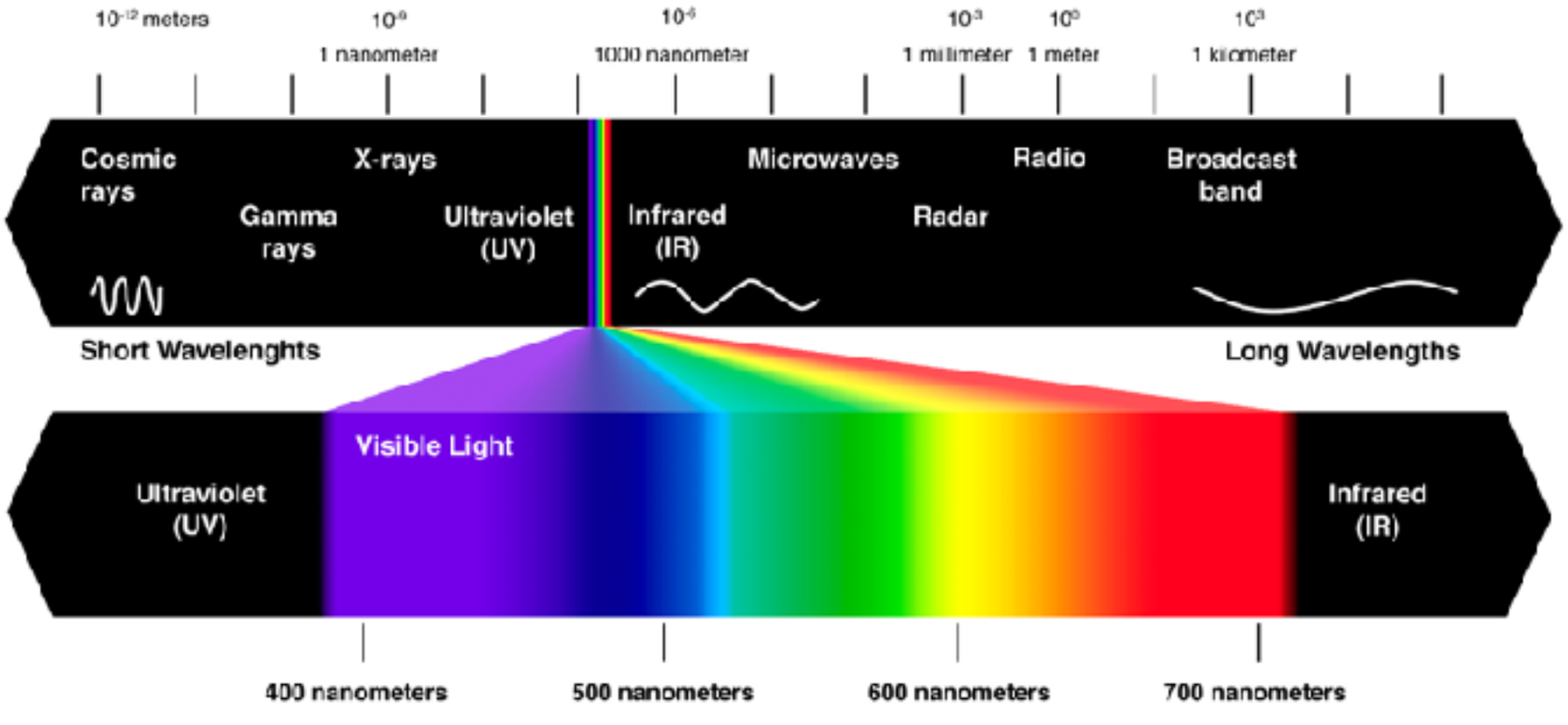
image

What is Computer Graphics?

- Based on:
  - Geometry
  - Physics
  - Physiology/Neurology/Psychology
- with a lot of simplifications and hacks to make it tractable and look good.

Physics of light

- Light is an electromagnetic wave, the same as radio waves, microwaves, X-rays, etc.
- The visible spectrum (for humans) consists of waves with wavelength between 400 and 700 nanometers.



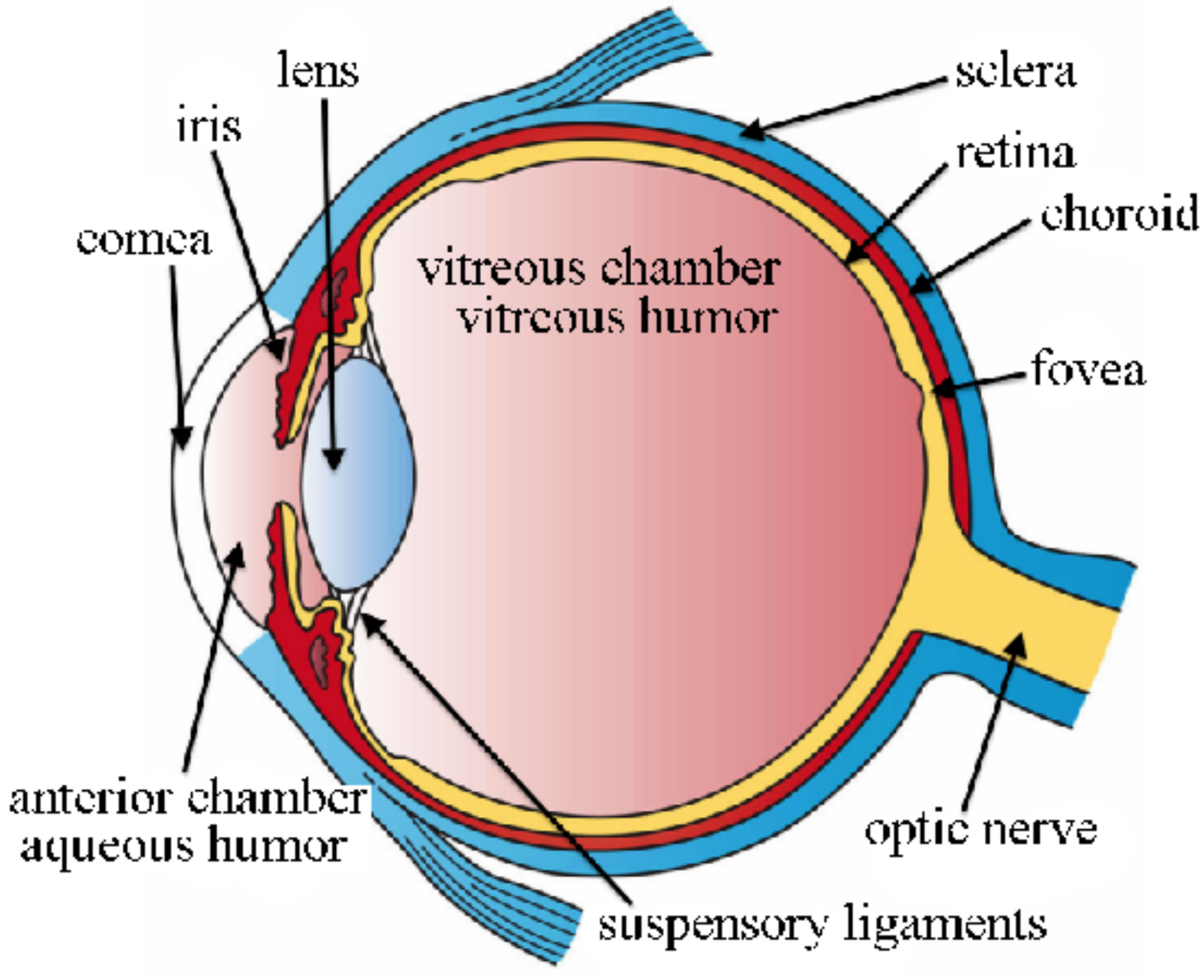
Non-spectral colours

Some light sources, such as lasers, emit light of essentially a single wavelength or “pure spectral” light (red, violet and colors of the rainbow).

Other colours (e.g. white, purple, pink,brown) are non-spectral.

There is no single wavelength for these colours, rather they are mixtures of light of different wavelengths.

The Eye



http://open.umich.edu/education/med/resources/second-look-series/materials

Colour perception

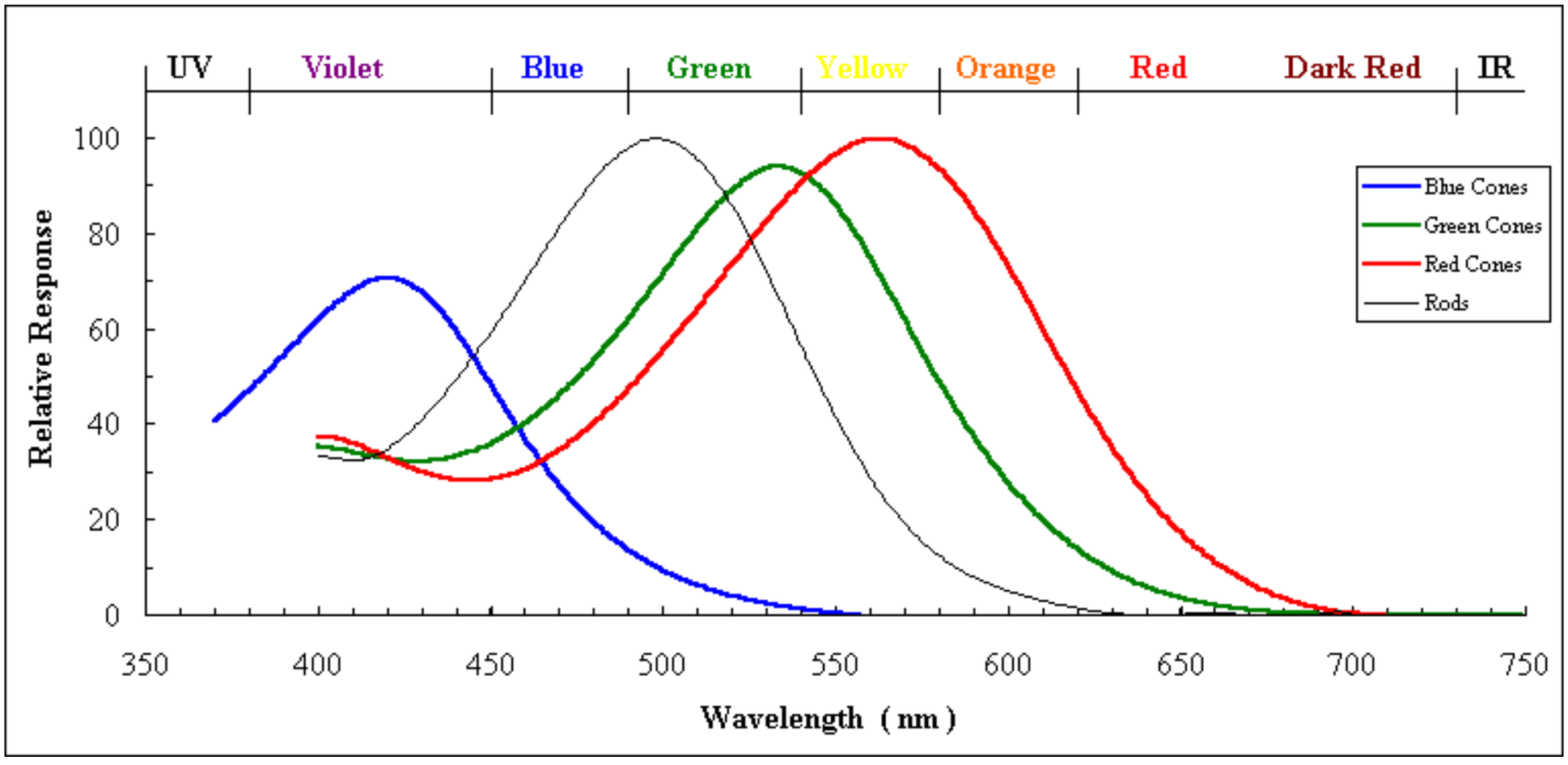
- The retina (back of the eye) has two different kinds of photoreceptor cells: rods and cones.

- Rods are good at handling low-level lighting (e.g. moonlight). They do not detect different colours and are poor at distinguishing detail.

- Cones respond better in brighter light levels. They are better at discerning detail and colour.

Tristimulus Theory

- Most people have three different kinds of cones which are sensitive to different wavelengths.



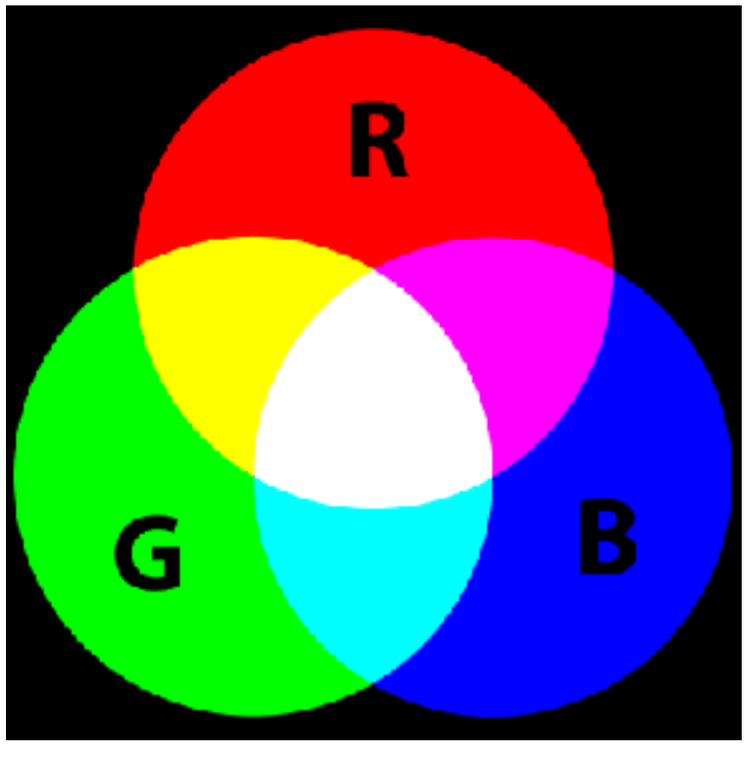
Colour blending

- As a result of this, different mixtures of light will appear to have the same colour, because they stimulate the cones in the same way.

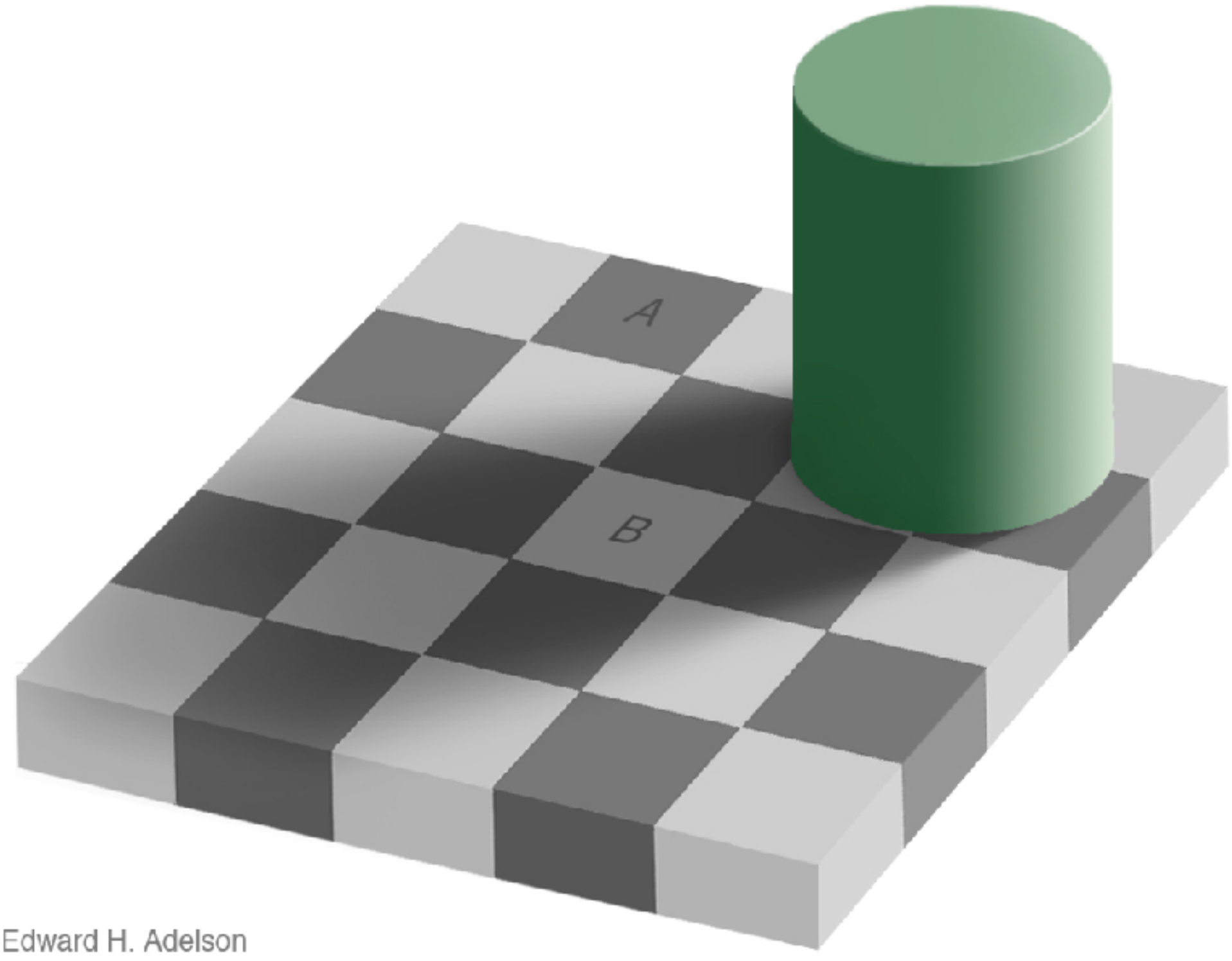
- For example, a mixture of red and green light will appear to be yellow.

Colour blending

- We can take advantage of this in a computer by having monitors with only red, blue and green phosphors in pixels.
- Other colours are made by mixing these lights together.

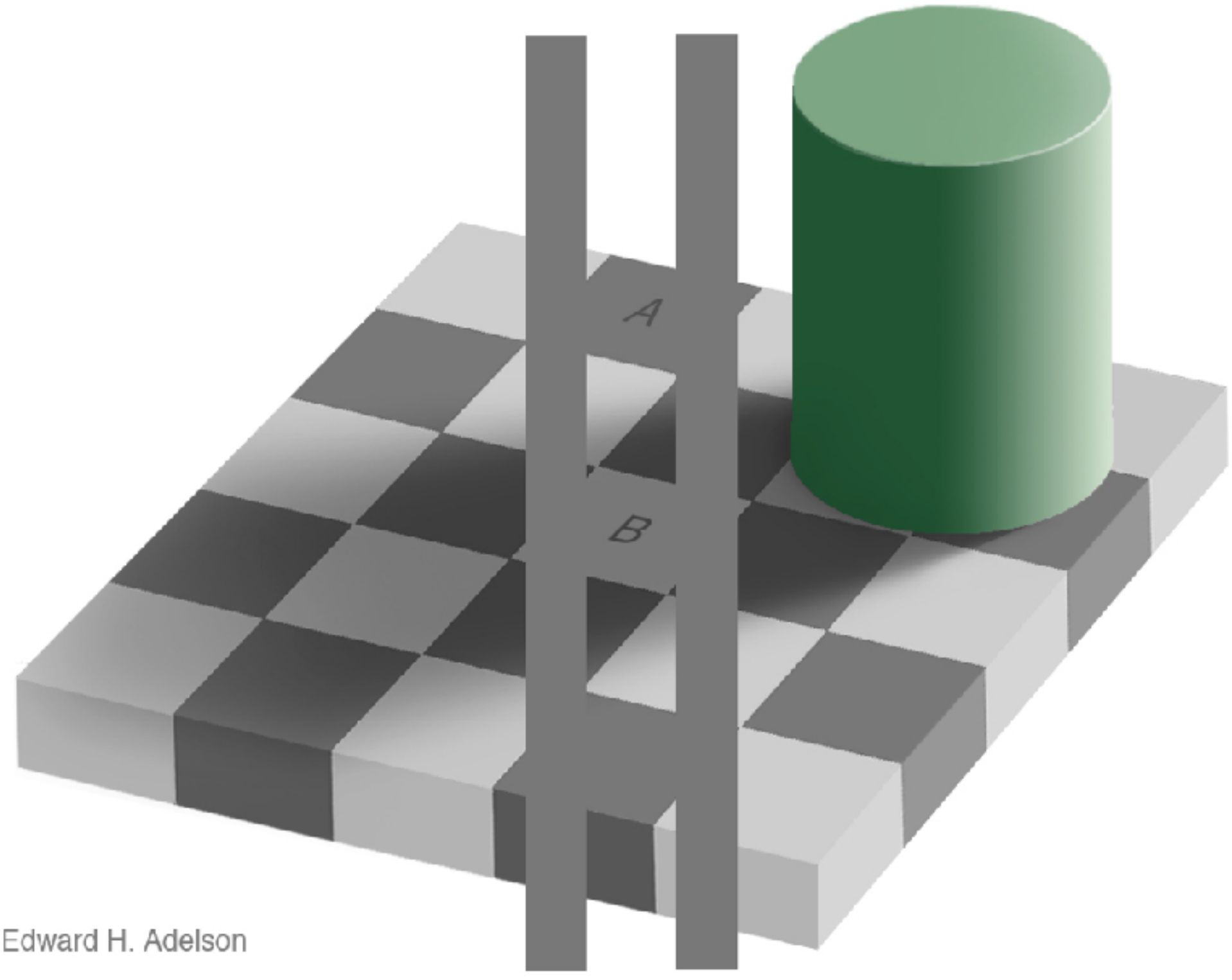


Checker Shadow Illusion



Edward H. Adelson

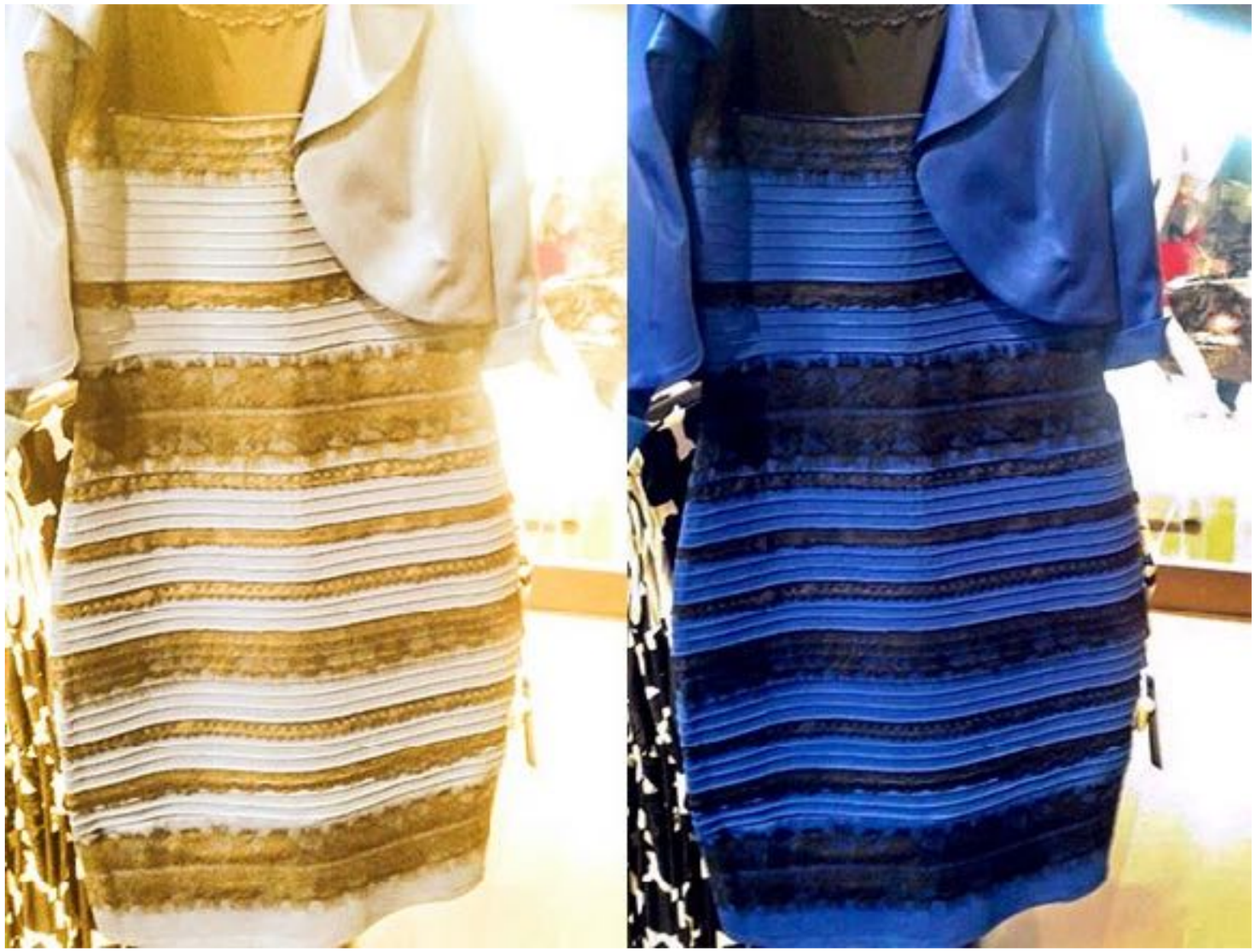
Checker Shadow Illusion



Edward H. Adelson

Color Illusions

Color Illusions



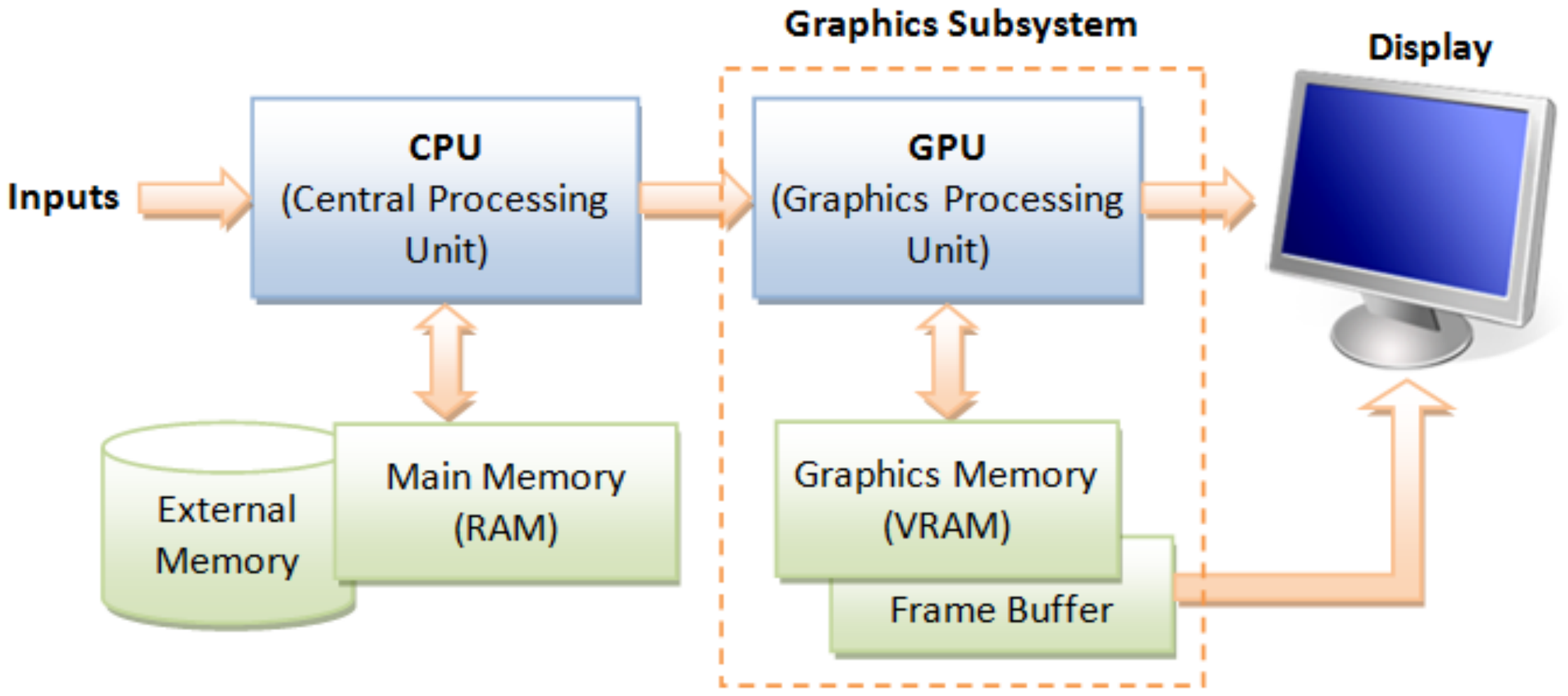
Images

- A 2D array of pixels
  - Each pixel has a red, green and blue value (RGB).
- The output of the graphics pipeline
- Animation is just rendering many images quickly one after the other
  - Usually 30 or 60 images (or frames) a second
- Interactive graphics applications (e.g. Games) generate frames in response to user input

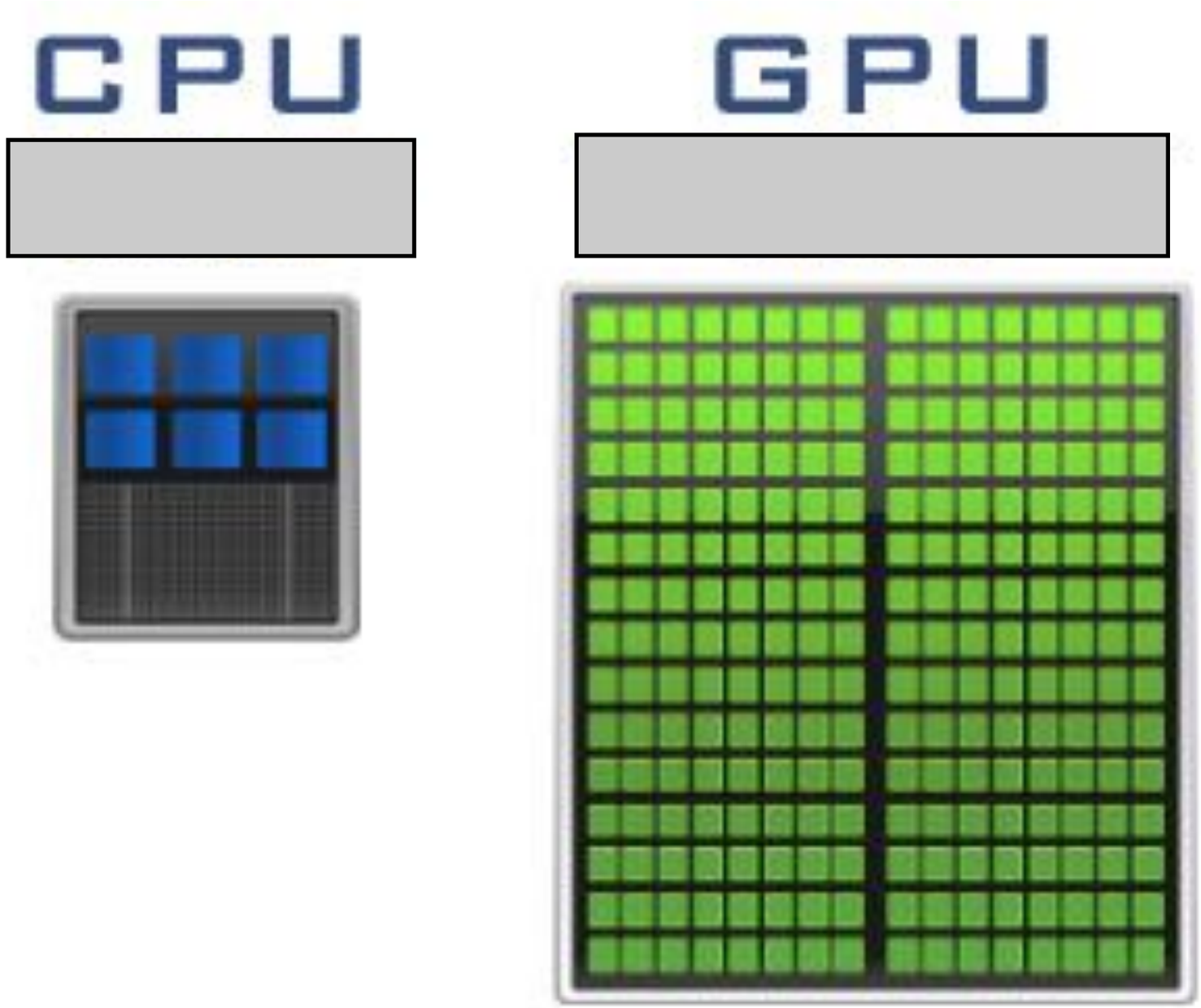
Realistic rendering

- Our main focus will be on realistic rendering of 3D models. i.e. Simulating a photographic image from a camera.
- Note however: most art is not realistic but involves some kind of abstraction.
- Realism is easier because physics is more predictable than psychology.
- The same techniques that are used to create realism can also be applied to more abstract rendering though

Hardware


Graphics Subsystem
Display
Inputs
CPU
(Central Processing Unit)
GPU
(Graphics Processing Unit)
External Memory
Main Memory (RAM)
Graphics Memory (VRAM)
Frame Buffer

CPU vs GPU



CPU vs GPU

- CPU consists of a few cores optimized for sequential serial processing
- GPU has a massively parallel architecture (SIMD/Single Instruction Multiple Data) consisting of smaller special purpose cores designed for parallel work.

OpenGL

- A low-level 2D/3D graphics API.
  - Free, Open source
  - Cross platform (incl. web and mobile)
  - Highly optimised
  - Designed to use special purpose hardware (GPU)
  - We will be using OpenGL

DirectX

- Direct3D
  - Microsoft proprietary
  - Only on MS platforms or through emulation (Wine, VMWare)
  - Roughly equivalent features

Vulcan

- Next generation graphics API
  - Still fairly new
  - Only limited support on some platforms (e.g. Mac)
  - Not quite ready for teaching yet, but hopefully soon

Do it yourself

- Generally a bad idea:
  - Reinventing the wheel
  - Numerical accuracy is hard
  - Efficiency is also hard
  - Hardware variations

Low-level graphics

- OpenGL is used to:
  - transfer data to the graphics memory
  - draw primitive shapes (points, lines, triangles, …) using that data
- More complex things like curves, composite shapes, etc. we have to implement ourselves
  - Composing primitives
  - Running programs (shaders) on the GPU

High-level graphics

- Numerous ways
- Unity
- Game engines
- Microsoft Paint?

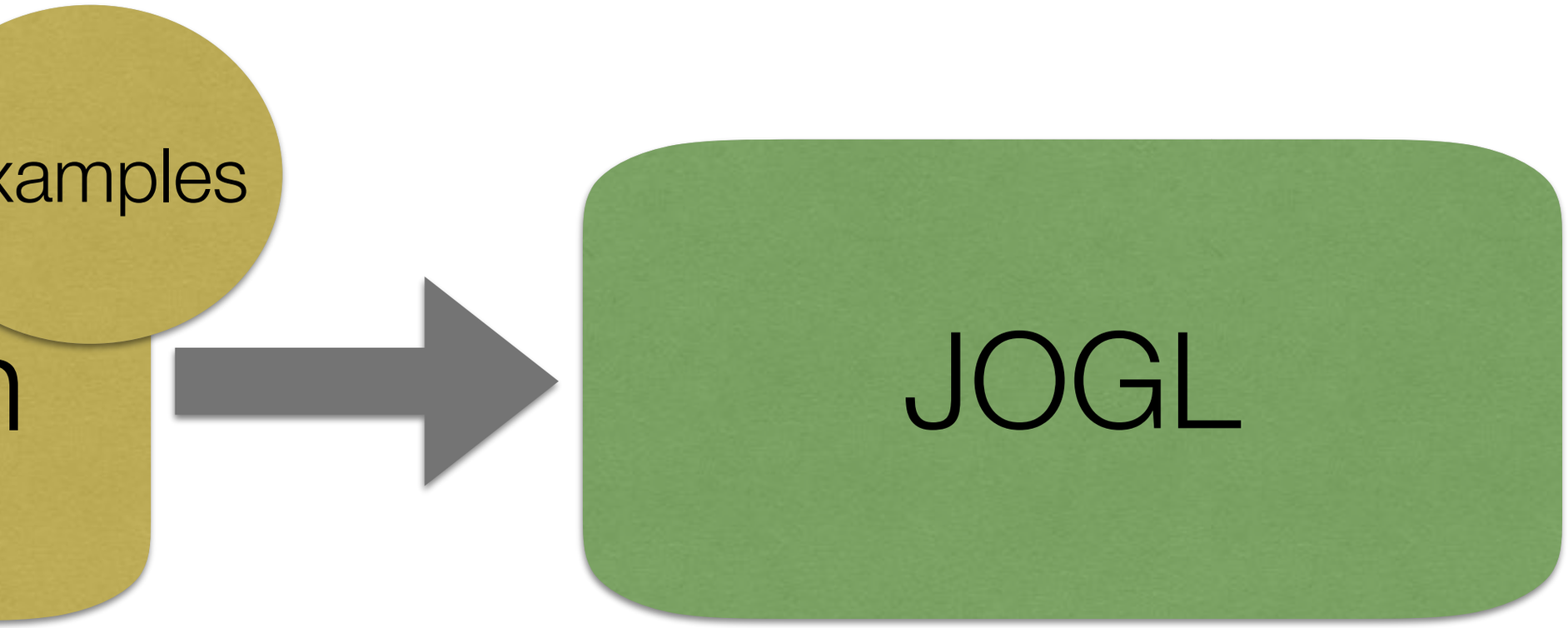
The plan

- Learn about techniques, concepts and algorithms relating to computer graphics.
- Use them to implement a high-level graphics library
  - In lectures, tutes, assignments
  - Using OpenGL for the low-level components

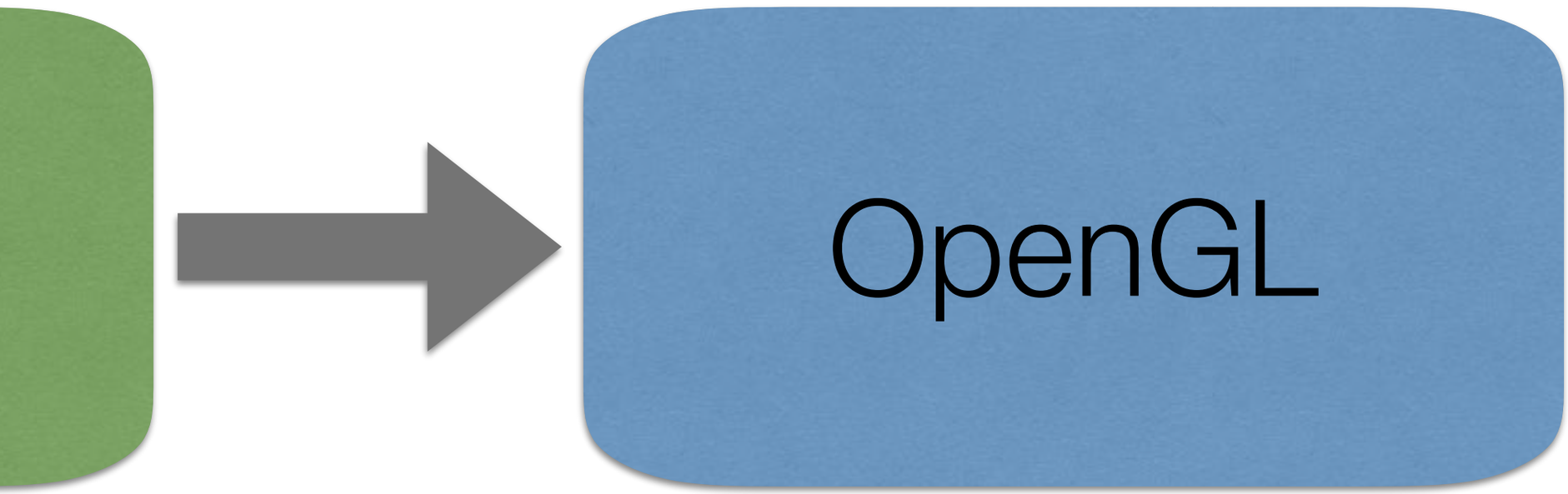
UNSWgraph

Examples

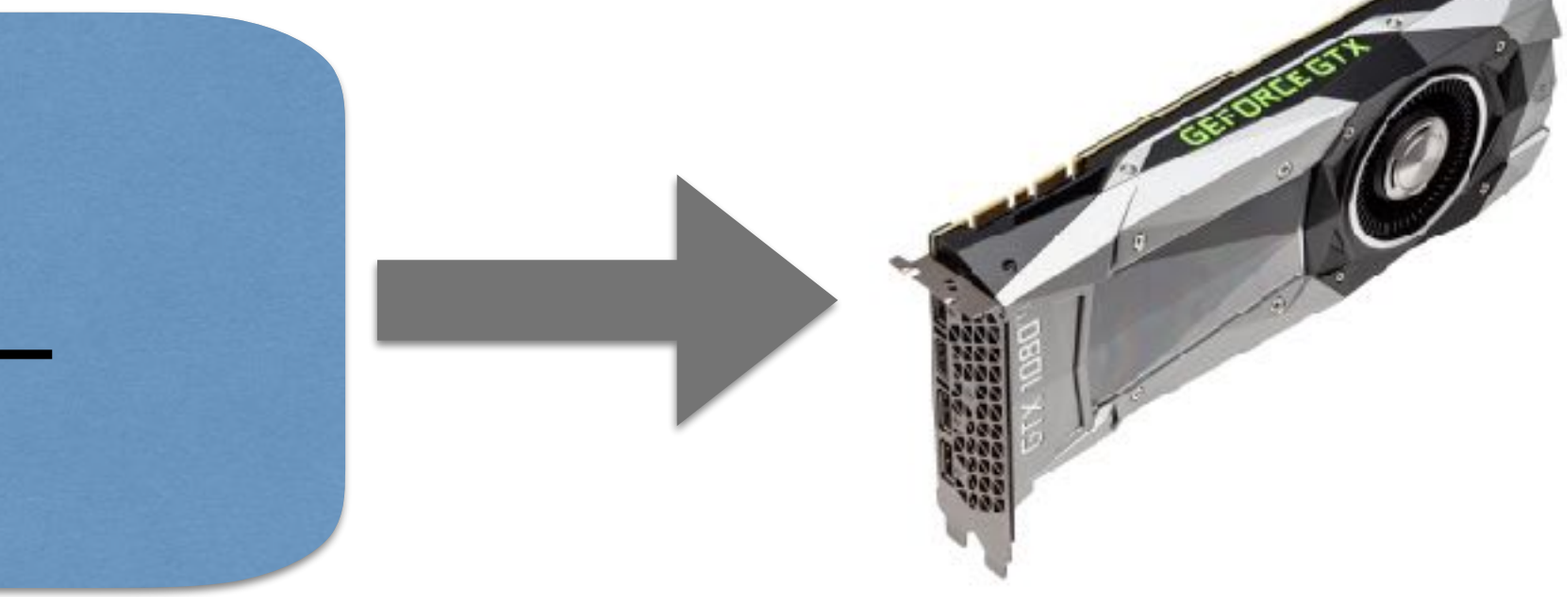
- A small high-level graphics library
  - Only VERY basic features (week 1)
  - We will explore and extend it throughout the course
  - Contains some example programs



- A Java library
- A wrapper around OpenGL (a C library)
- Contains NEWT, a basic windowing toolkit
- http://jogamp.org/jogl/www/

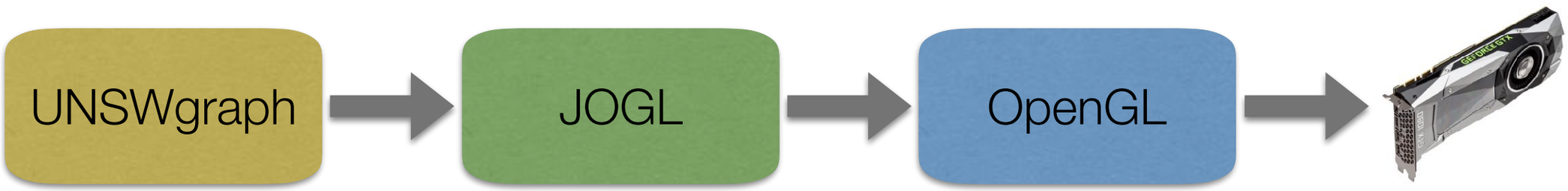


- Implementation of the API provided by the GPU driver
- We don’t *know* how it works internally



- For this course we will focus on how to use it, not the hardware architecture

Pipeline



UNSWgraph

- The lab contains instructions for setting up UNSWgraph and running an example program.
- Short version: It is packaged as an eclipse project, so can be directly imported into eclipse with minimal hassle
- NOTE: Doesn’t work on VLAB

My first graphics program

- See HelloDot.java
- Shows ALL features of UNSWgraph version 0.1

Application

- Applications have a single NEWT window
- 2D applications give a simple 2D canvas to draw on.
- The size of the window is given to the constructor.
- We can also set the background color.

```java
public class HelloDot extends Application2D {

    public HelloDot() {
        super("HelloDot", 600, 600);
        this.setBackground(new Color(1f, 1f, 1f));
    }

    public static void main(String[] args) {
        HelloDot example = new HelloDot();
        example.start();
    }

    @Override
    public void display(GL3 gl) {
        super.display(gl);
        Point2D point = new Point2D(0f, 0f);
        point.draw(gl);
    }

}
```

window size

background colour

RGB

- Colors are defined using Red (R), Green (G), Blue (B).
- R,G,B values range from 0.0 (none) to 1.0 (full intensity)

```
public class HelloDot extends Application2D {

    public HelloDot() {
        super("HelloDot", 600, 600);
        this.setBackground(new Color(1f, 1f, 1f));
    }

    public static void main(String[] args) {
        HelloDot example = new HelloDot();
        example.start();
    }

    @Override
    public void display(GL3 gl) {
        super.display(gl);
        Point2D point = new Point2D(0f, 0f);
        point.draw(gl);
    }

}
```

window size

background colour

display handler

Event-based Programming

- UNSWgraph and NEWT are event-driven.
- This requires a different approach to procedural programming:
  - The main body sets up the components and registers event handlers, then quits.
  - Events are dispatched by the event loop.
  - Handlers are called when events occur.
    - e.g. display() is called 60 times a second

```
public class HelloDot extends Application2D {

    public HelloDot() {
        super("HelloDot", 600, 600);
        this.setBackground(new Color(1f, 1f, 1f));
    }

    public static void main(String[] args) {
        HelloDot example = new HelloDot();
        example.start();
    }

    @Override
    public void display(GL3 gl) {
        super.display(gl);
        Point2D point = new Point2D(0f, 0f);
        point.draw(gl);
    }

}
```

window size

background colour

display handler

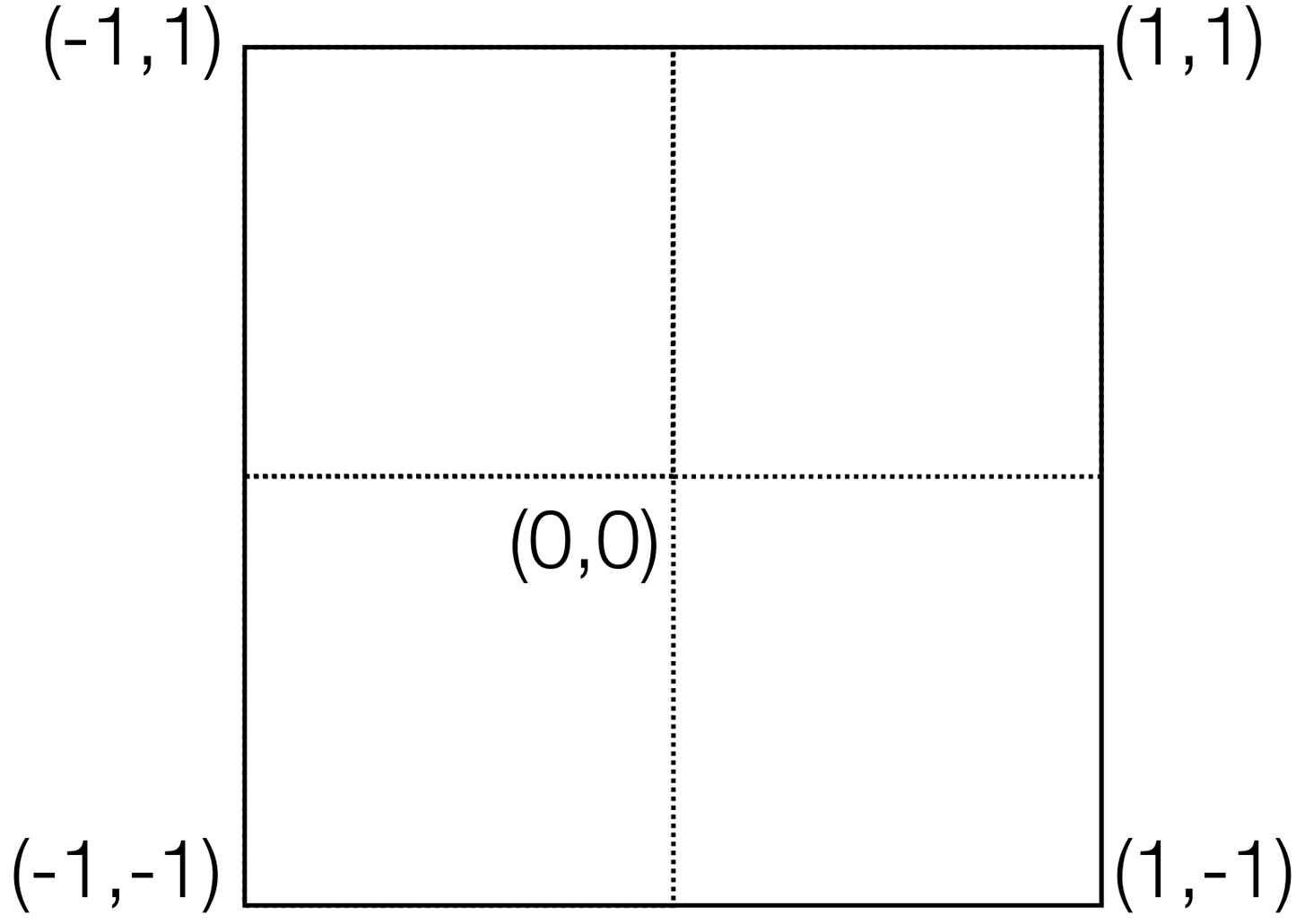
point position

Viewport

- We talk in general about the viewport as the piece of the screen we are drawing on.
- It may be a window, part of a window, or the whole screen. (In UNSWgraph by default it is the whole window – minus the border)
- It can be any size but we assume it is always a rectangle.
- It has its own coordinate system

Coordinate system

- By default the viewport is centred at (0,0). The left boundary is at x=-1, the right at x=1, the bottom at y=-1 and the top at y=1.



But what's really going on?

- See Point2D.draw()
- In the draw method for point we have to do 4 main things
  - Create a buffer in main memory containing the point coordinates
  - Transfer that buffer to GPU memory
  - Tell the GPU to draw that buffer as a point
  - Free the buffer in GPU memory

GL3

- GL3 provides access to all the normal OpenGL methods and constants.
- http://jogamp.org/deployment/v2.2.4/javadoc/jogl/javadoc/javax/media/opengl/GL3.html
- A GL3 object can’t be constructed cloned or copied in any way
- We have to pass it through to the methods that need it

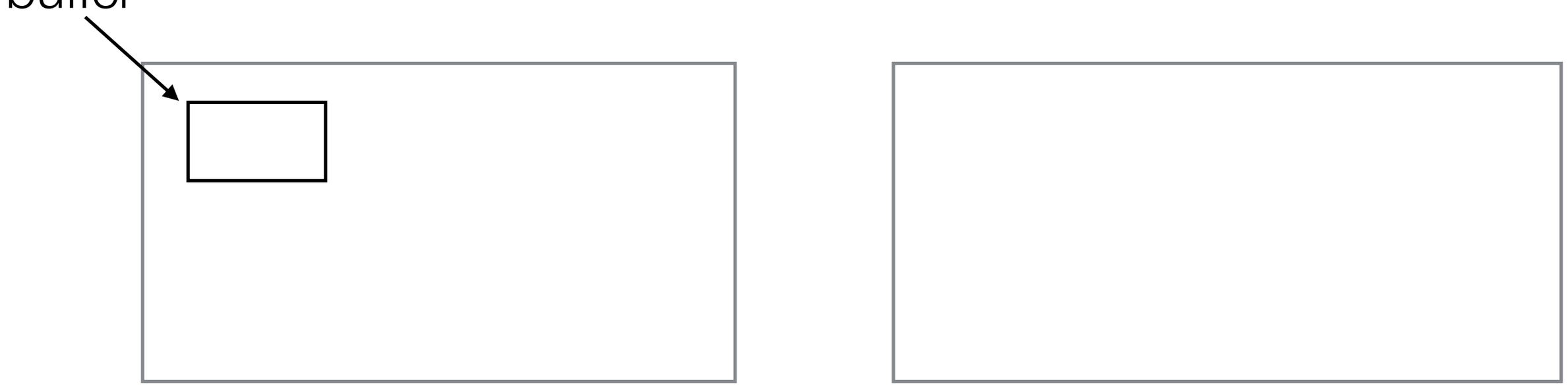
We have two memory spaces

Main Memory

GPU Memory

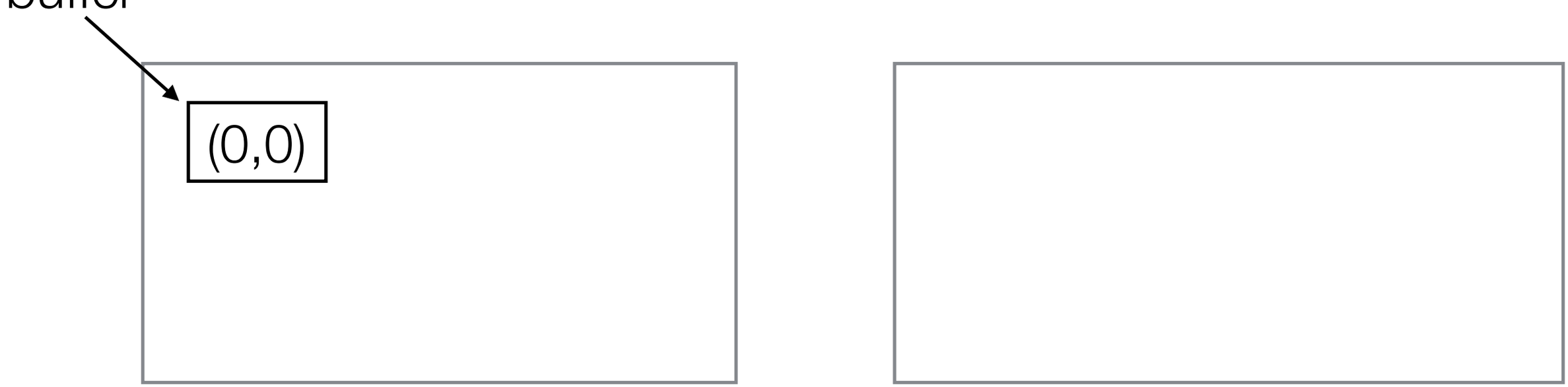
```
Point2DBuffer buffer = new Point2DBuffer(1);
```

Create a buffer that can store 1 point
The buffer is pinned in main memory.



```
buffer.put(0, this);
```

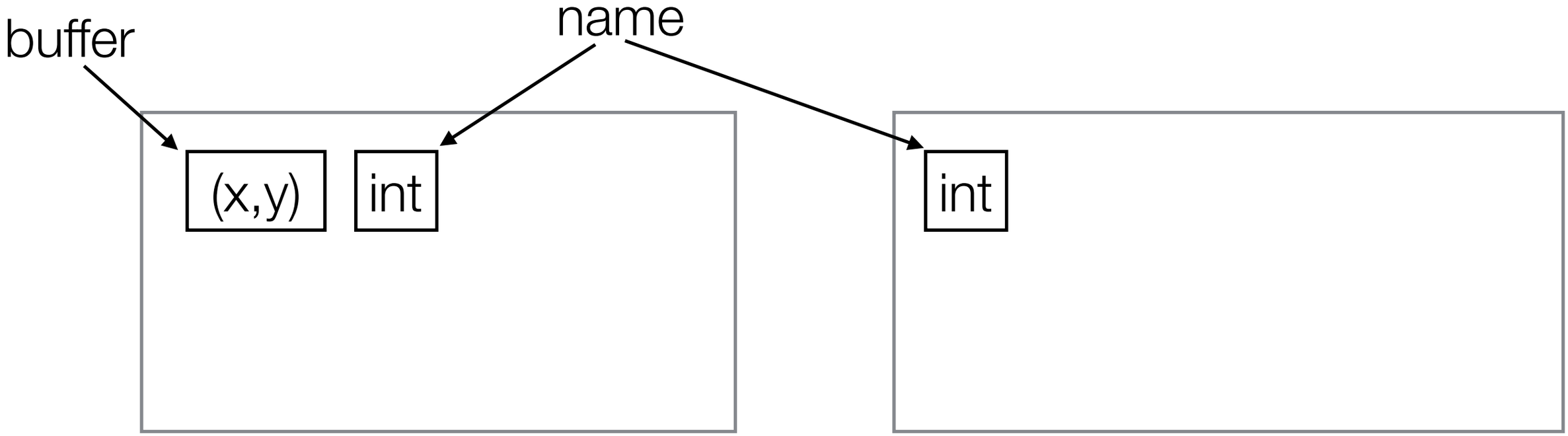
Store the value of this point at index 0 in the buffer



http://docs.gl/gl3/glGenBuffers

```
int[] names = new int[1];
gl.glGenBuffers(1, names, 0);
```

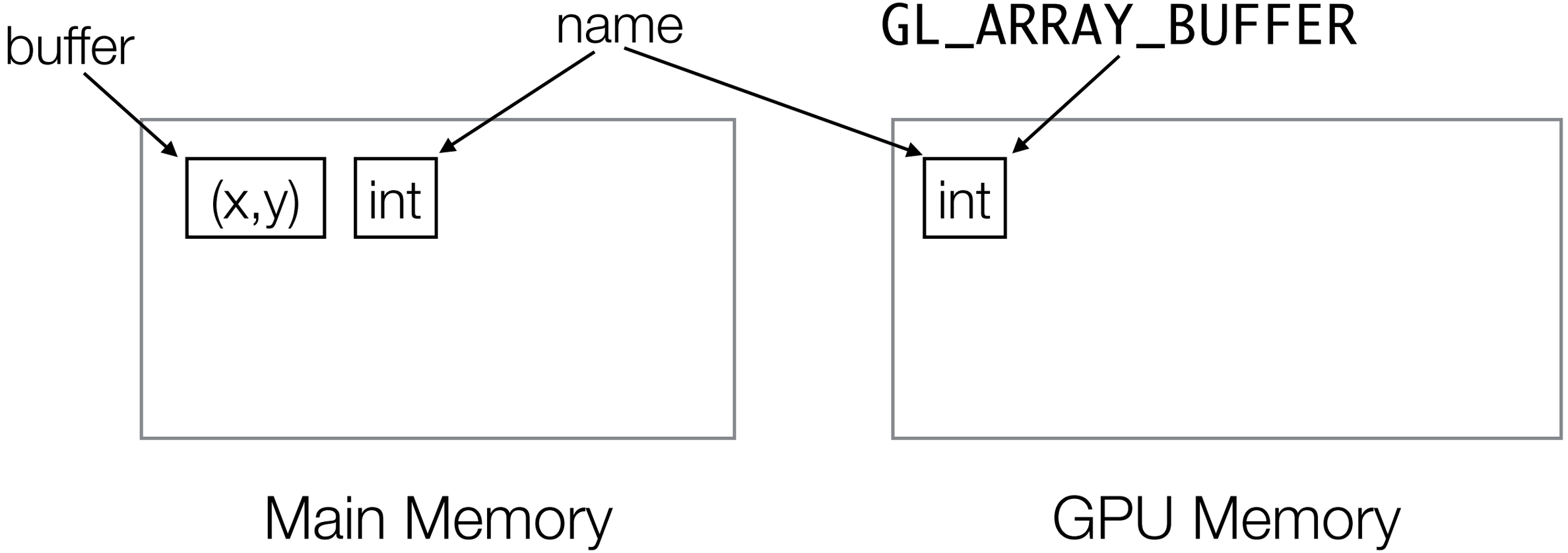
Create a new name for a buffer



http://docs.gl/gl3/glBindBuffer

```
gl.glBindBuffer(GL.GL_ARRAY_BUFFER, names[0]);
```

This is the buffer we want to use. All future buffer operations will be on this buffer.



```
void glBindBuffer(int target,  // Binding target
                  int buffer); // Name of buffer
```

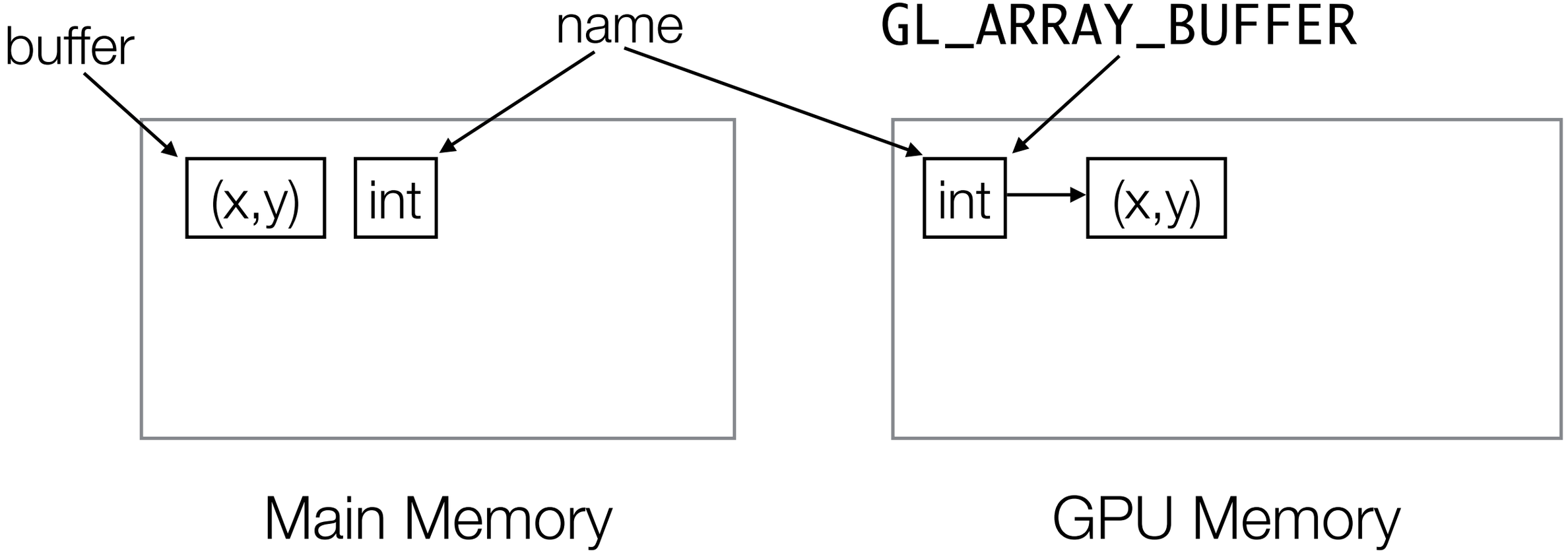
Buffer targets

- OpenGL can only have one active buffer of a particular target
- Binding a buffer to GL_ARRAY_BUFFER tells OpenGL that all future operations on the GL_ARRAY_BUFFER are for this buffer
- The GL_ARRAY_BUFFER target is a general purpose target
- Other buffer targets we will see in later weeks.

http://docs.gl/gl3/glBufferData

```
gl.glBufferData(GL.GL_ARRAY_BUFFER, 2 * Float.BYTES,
    buffer.getBuffer(), GL.GL_STATIC_DRAW);
```

This allocates the buffer in graphics memory and transfers the data from main memory into it



```
void glBufferData(
    int target,    // Destination
long size,         // Transfer size (in bytes)
Buffer data,       // Source
int usage);        // How it is used
```

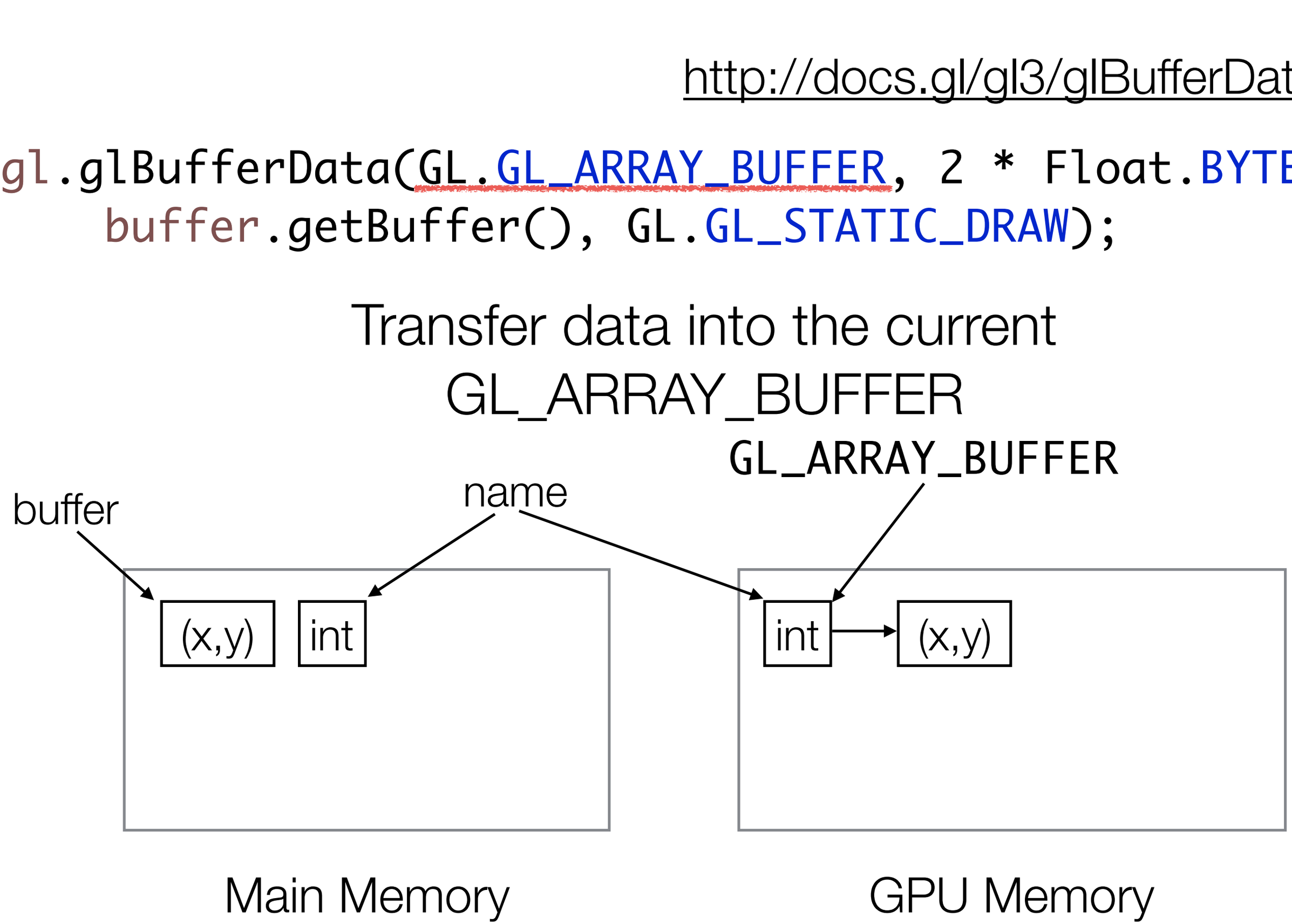
Buffer usage hints

- When allocating a buffer OpenGL lets you give a hint how it might be used.
- OpenGL is free to ignore this information but may use it to optimise how and where it stores the data.
- The most common hints are:
  - GL_STATIC_DRAW — Data will be modified once and used many times
  - GL_DYNAMIC_DRAW —Data will be modified repeatedly and used repeatedly

http://docs.gl/gl3/glBufferData

```
gl.glBufferData(GL.GL_ARRAY_BUFFER, 2 * Float.BYTES,
    buffer.getBuffer(), GL.GL_STATIC_DRAW);
```

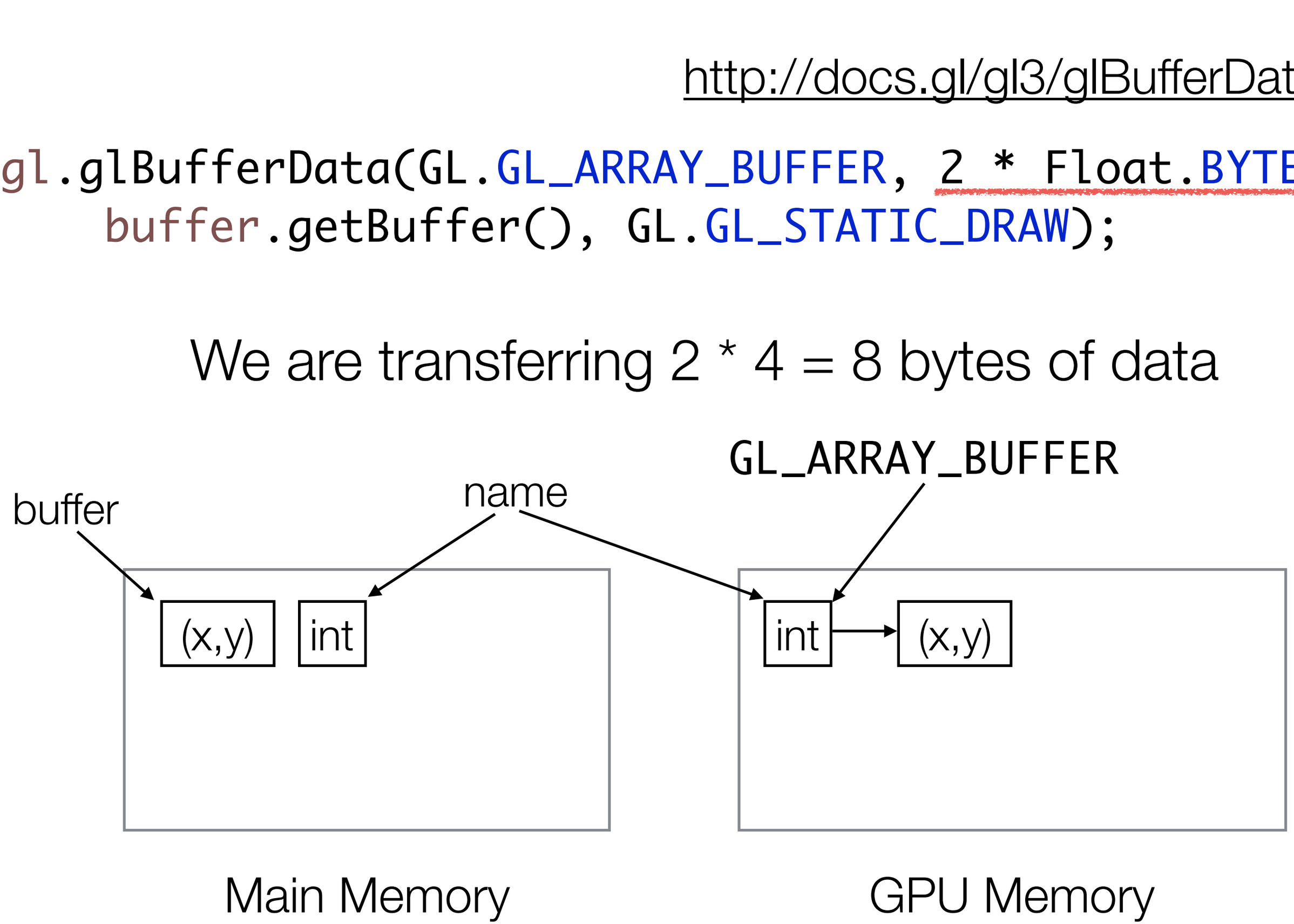
Transfer data into the current GL_ARRAY_BUFFER



http://docs.gl/gl3/glBufferData

```
gl.glBufferData(GL.GL_ARRAY_BUFFER, 2 * Float.BYTES,
    buffer.getBuffer(), GL.GL_STATIC_DRAW);
```

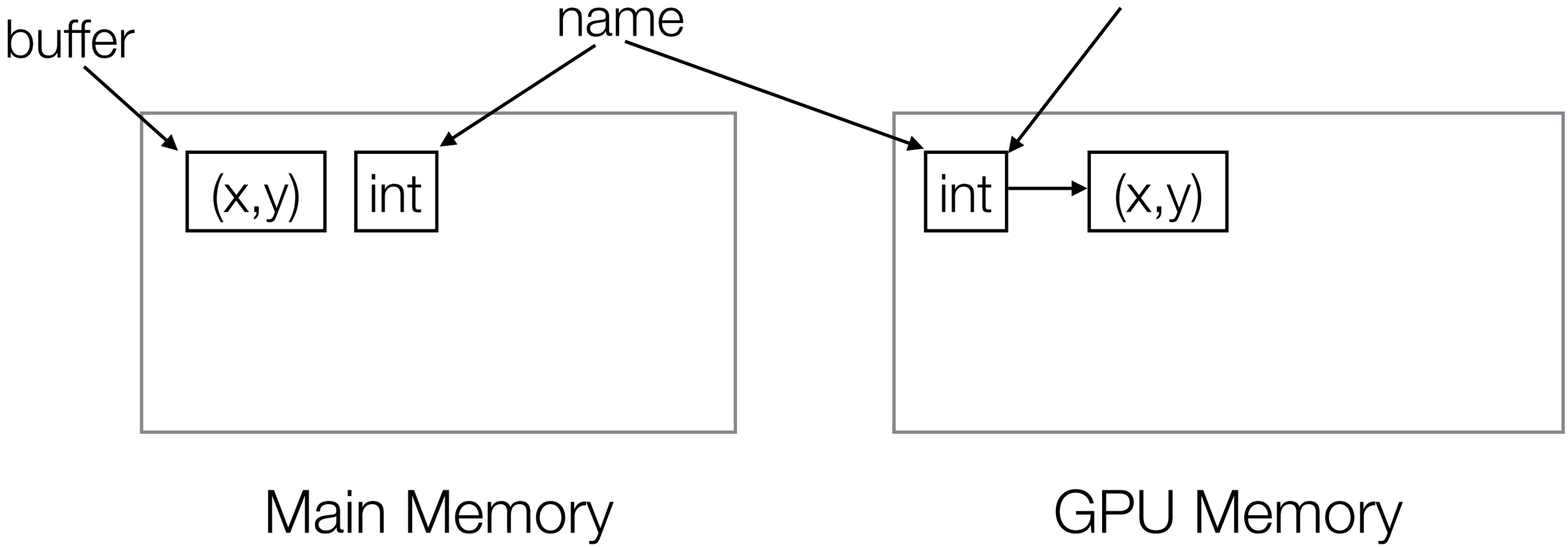
We are transferring 2 * 4 = 8 bytes of data



http://docs.gl/gl3/glBufferData

```
gl.glBufferData(GL.GL_ARRAY_BUFFER, 2 * Float.BYTES,
    buffer.getBuffer(), GL.GL_STATIC_DRAW);
```

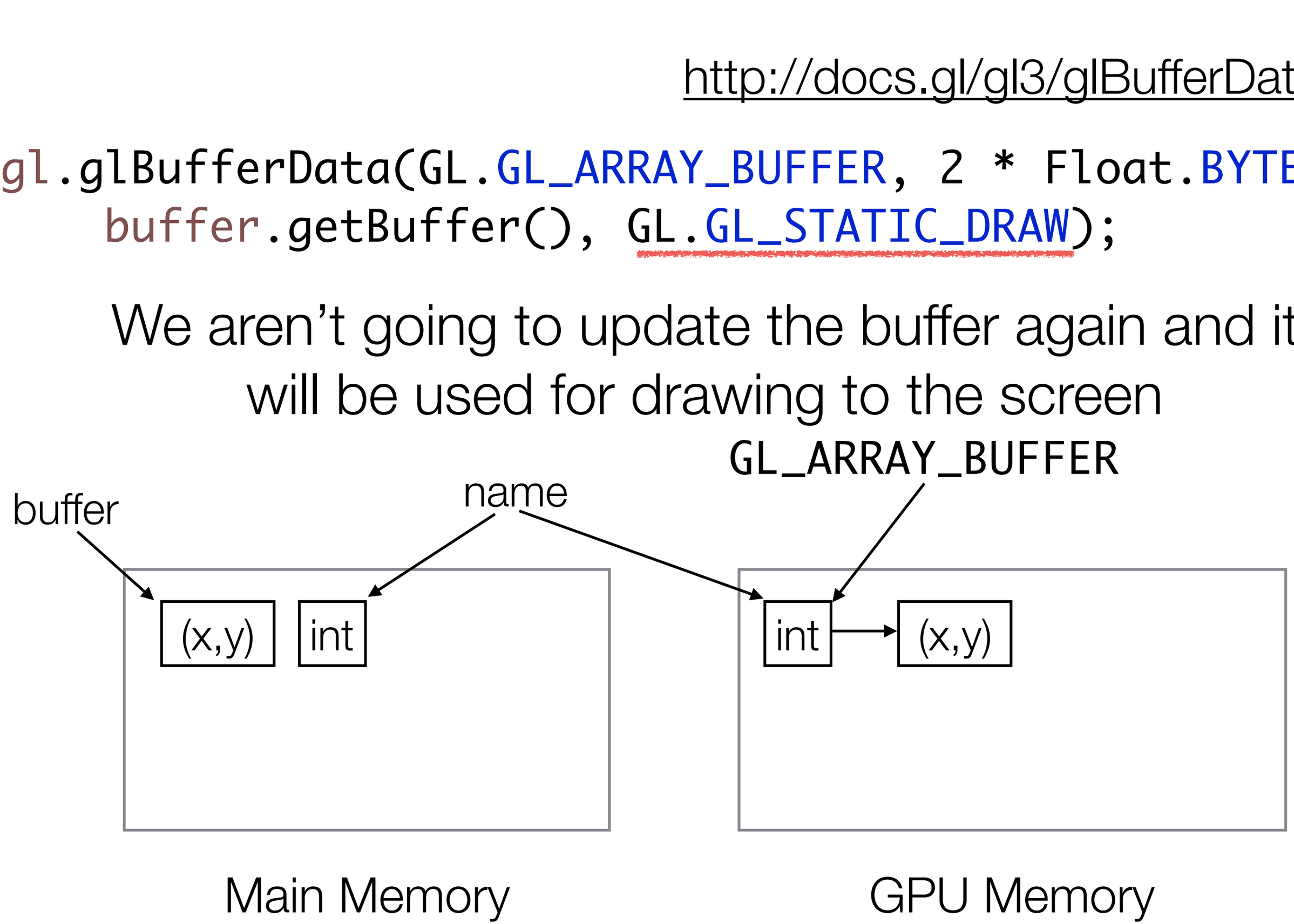
Using this buffer as a source



http://docs.gl/gl3/glBufferData

```
gl.glBufferData(GL.GL_ARRAY_BUFFER, 2 * Float.BYTES,
    buffer.getBuffer(), GL.GL_STATIC_DRAW);
```

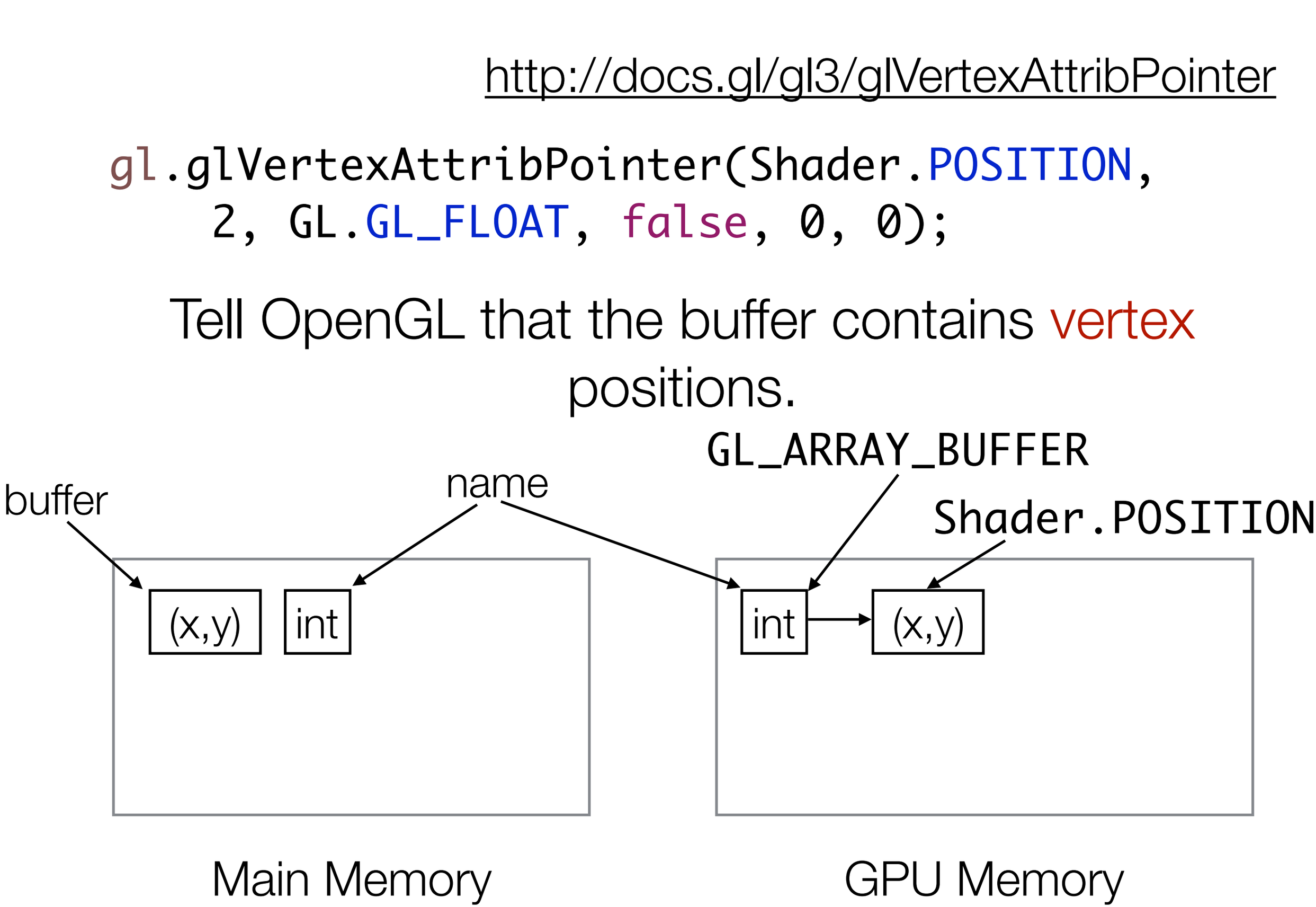
We aren't going to update the buffer again and it will be used for drawing to the screen



http://docs.gl/gl3/glVertexAttribPointer

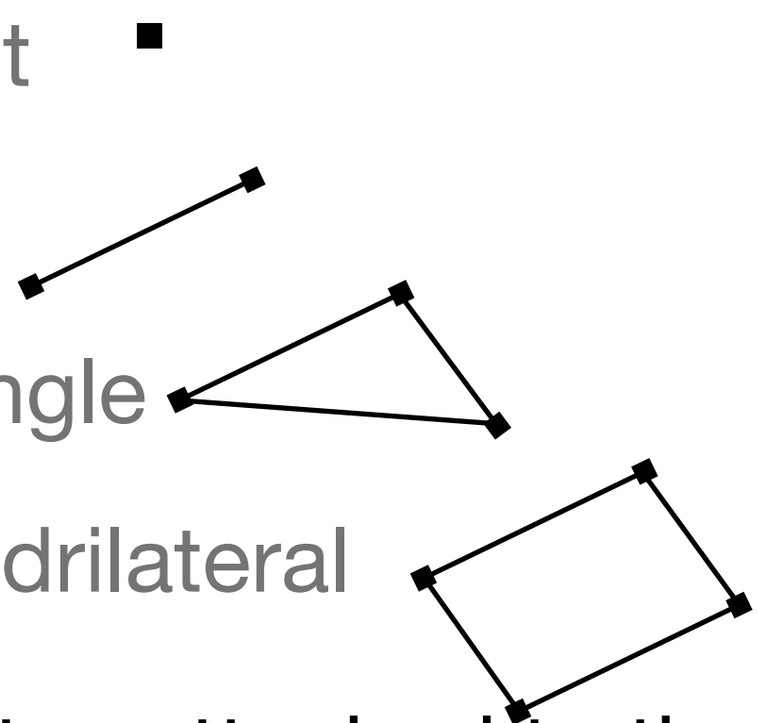
```
gl.glVertexAttribPointer(Shader.POSITION,
    2, GL.GL_FLOAT, false, 0, 0);
```

Tell OpenGL that the buffer contains vertex positions.



Vertex

- In OpenGL a vertex (plural: vertices) is a point that forms part of the definition of a geometric shape. For example:
  - 1 vertex defines a point
  - 2 vertices define a line
  - 3 vertices define a triangle
  - 4 vertices define a quadrilateral



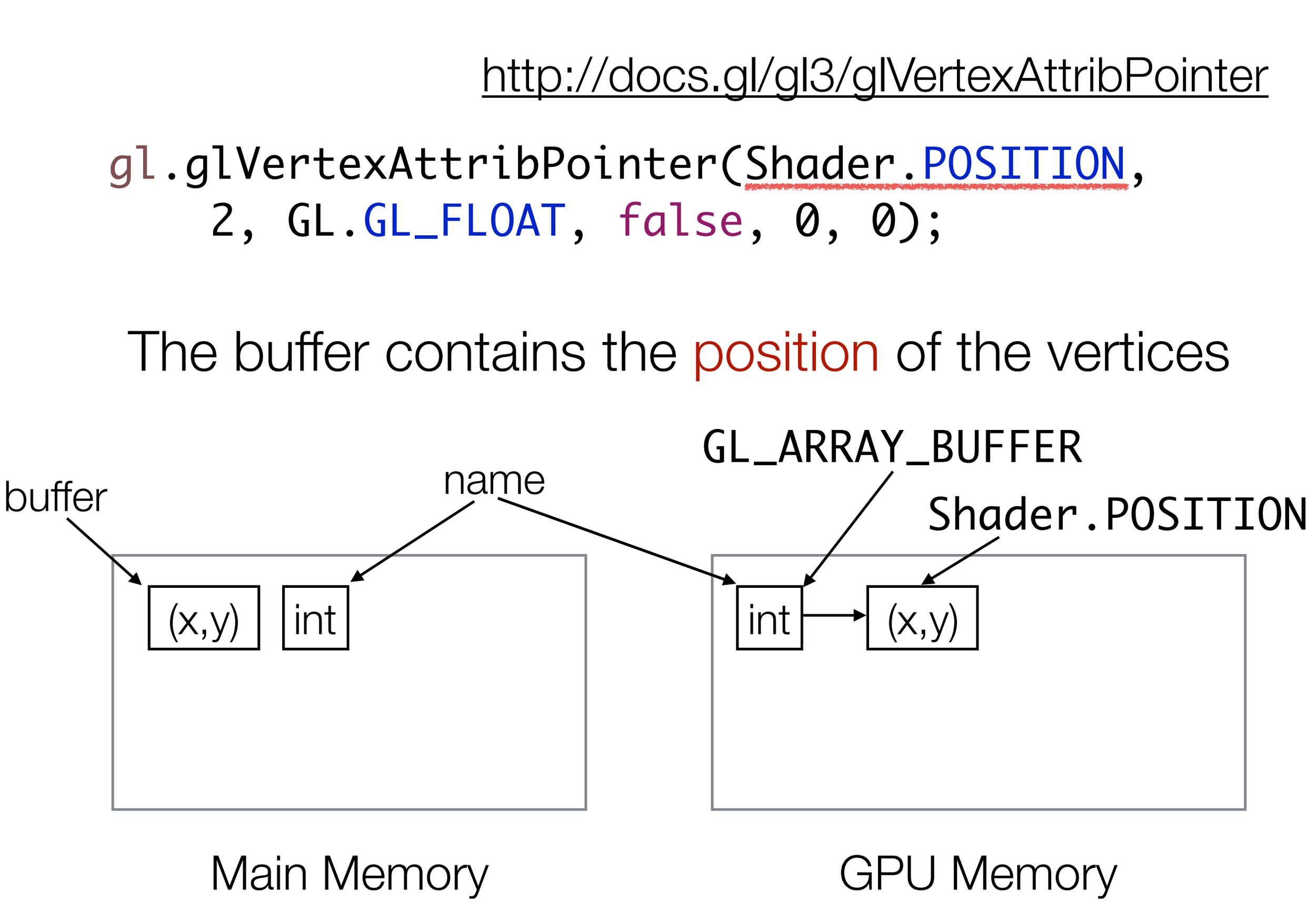
- Vertices can have attributes attached to them.

```
void glVertexAttribPointer(
    int index,                          // The attribute
    int size,                           // attribute size
    int type,                           // Primitive type
    boolean normalized,                 // Normalize ints
    int stride,                         // Padding
    long pointer_buffer_offset); // Start
```

http://docs.gl/gl3/glVertexAttribPointer

```
gl.glVertexAttribPointer(Shader.POSITION,
    2, GL.GL_FLOAT, false, 0, 0);
```

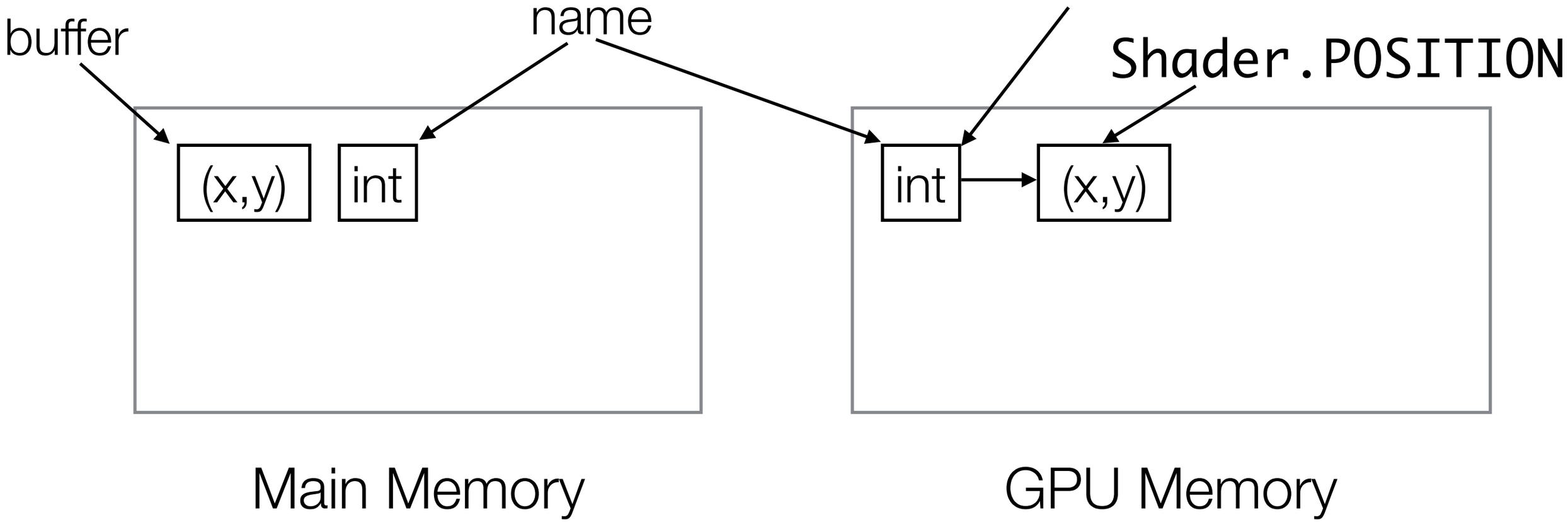
The buffer contains the position of the vertices



http://docs.gl/gl3/glVertexAttribPointer

```
gl.glVertexAttribPointer(Shader.POSITION,
    2, GL.GL_FLOAT, false, 0, 0);
```

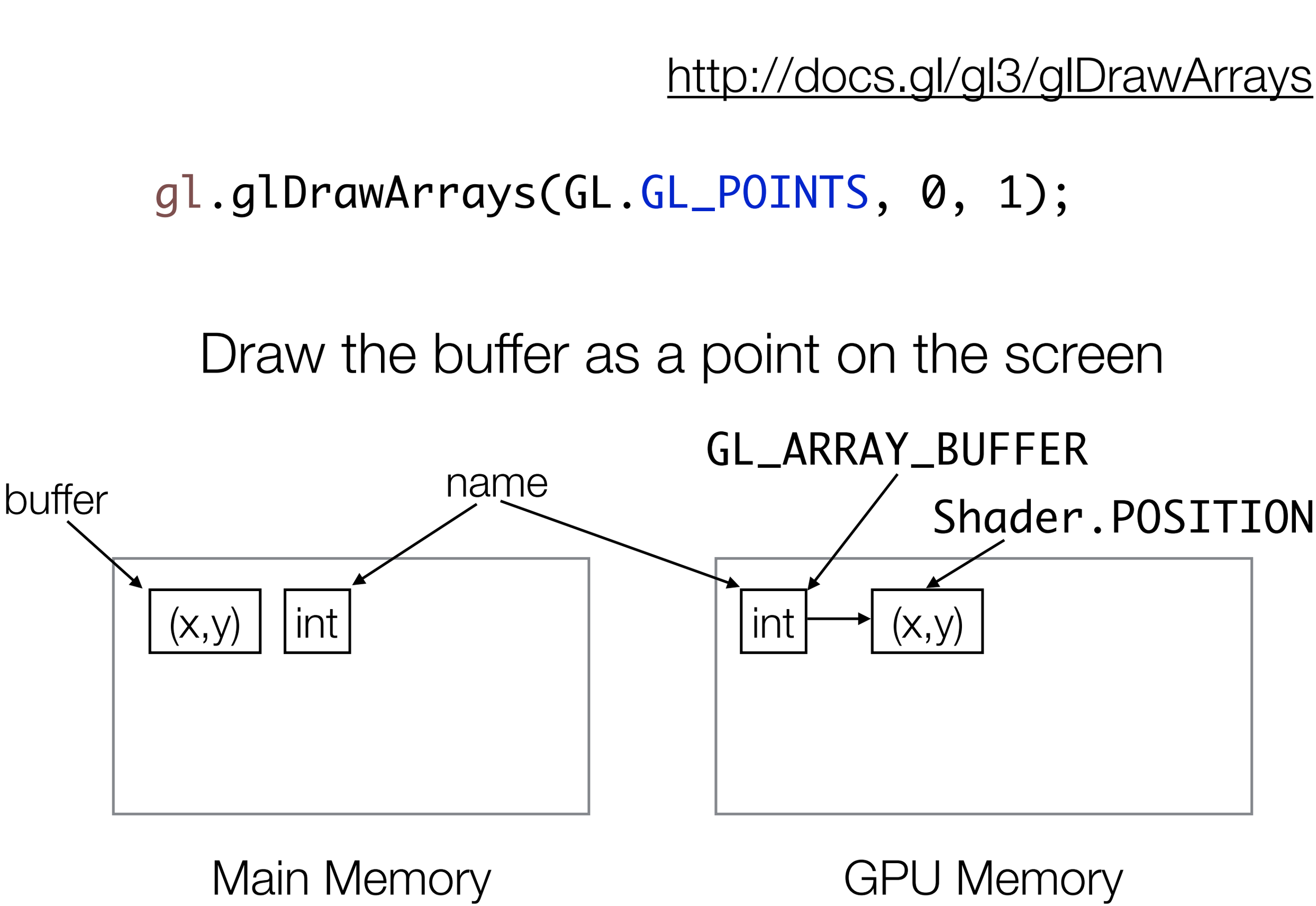
Each position has 2 floats associated with it.



http://docs.gl/gl3/glDrawArrays

```
gl.glDrawArrays(GL.GL_POINTS, 0, 1);
```

Draw the buffer as a point on the screen

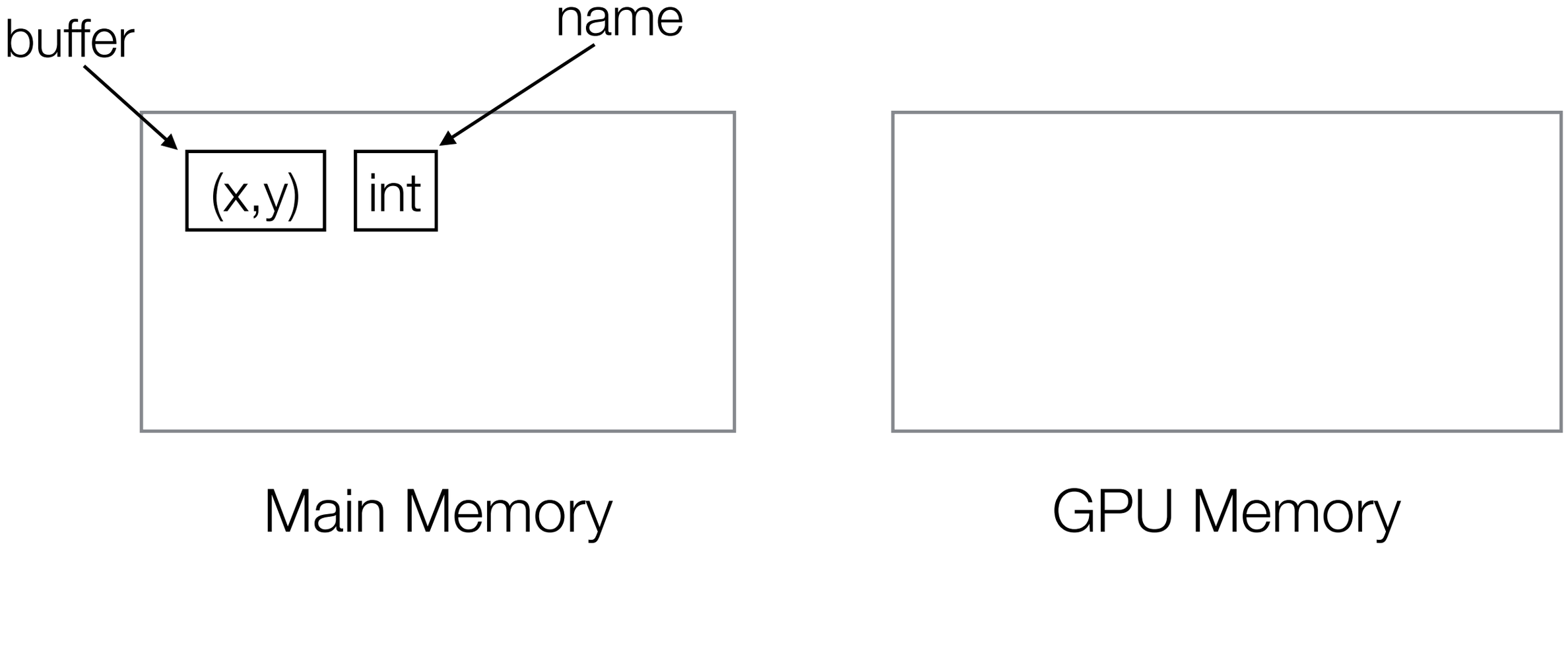


```
void glDrawArrays(int mode,   // Primitive to draw
                  int first,  // Starting vertex
                  int count); // Number of vertices
```

http://docs.gl/gl3/glDeleteBuffers

```
gl.glDeleteBuffers(1, names, 0);
```

Delete the buffer in graphics memory



```
void glDeleteBuffers(int n,
                     int[] buffers,
                     int buffers_offset);
```

OpenGL recap

- It is not Object-Oriented, despite us accessing it from Java
  - Use of ints instead of enums
  - Lots of effectively global state
- UNSWgraph is setup to try and report OpenGL errors, but in many cases failure is still silent (e.g. out of bounds errors)
- Error messages can be hard to decipher
- Need to rely on documentation

Questions

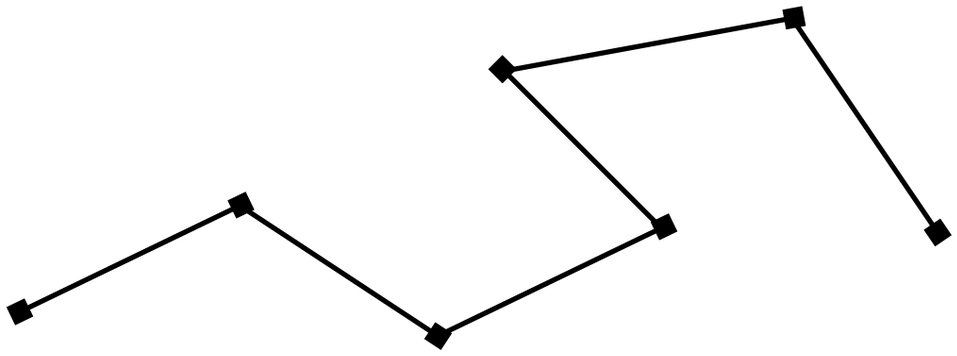
- What does it mean when we say OpenGL is low-level?
  - Hard to formally define what low-level is, but you should have intuition
- Can you remember all the arguments to glBufferData?
  - You can’t, and you shouldn’t.
  - References are really important (docs.gl)
- Isn’t programming like this really tedious?
  - Yes, but as experienced programmers we will quickly build up a codebase that makes it a lot easier

From points to lines

- See Line2D.java and HelloLine.java

Line strips

- A line strip is a series of points joined by lines



- They can be drawn with GL_LINE_STRIP
- See LineStrip2D.java

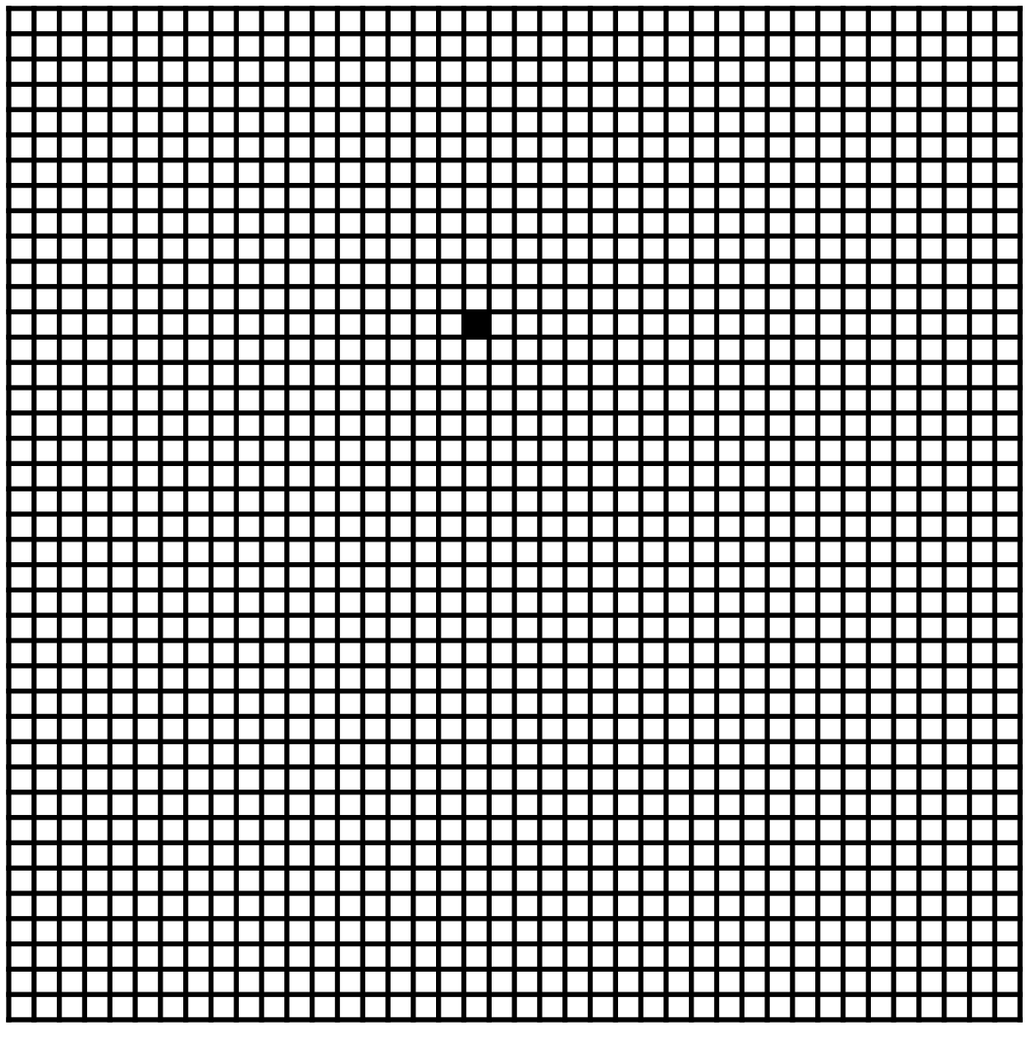
Mouse Input events

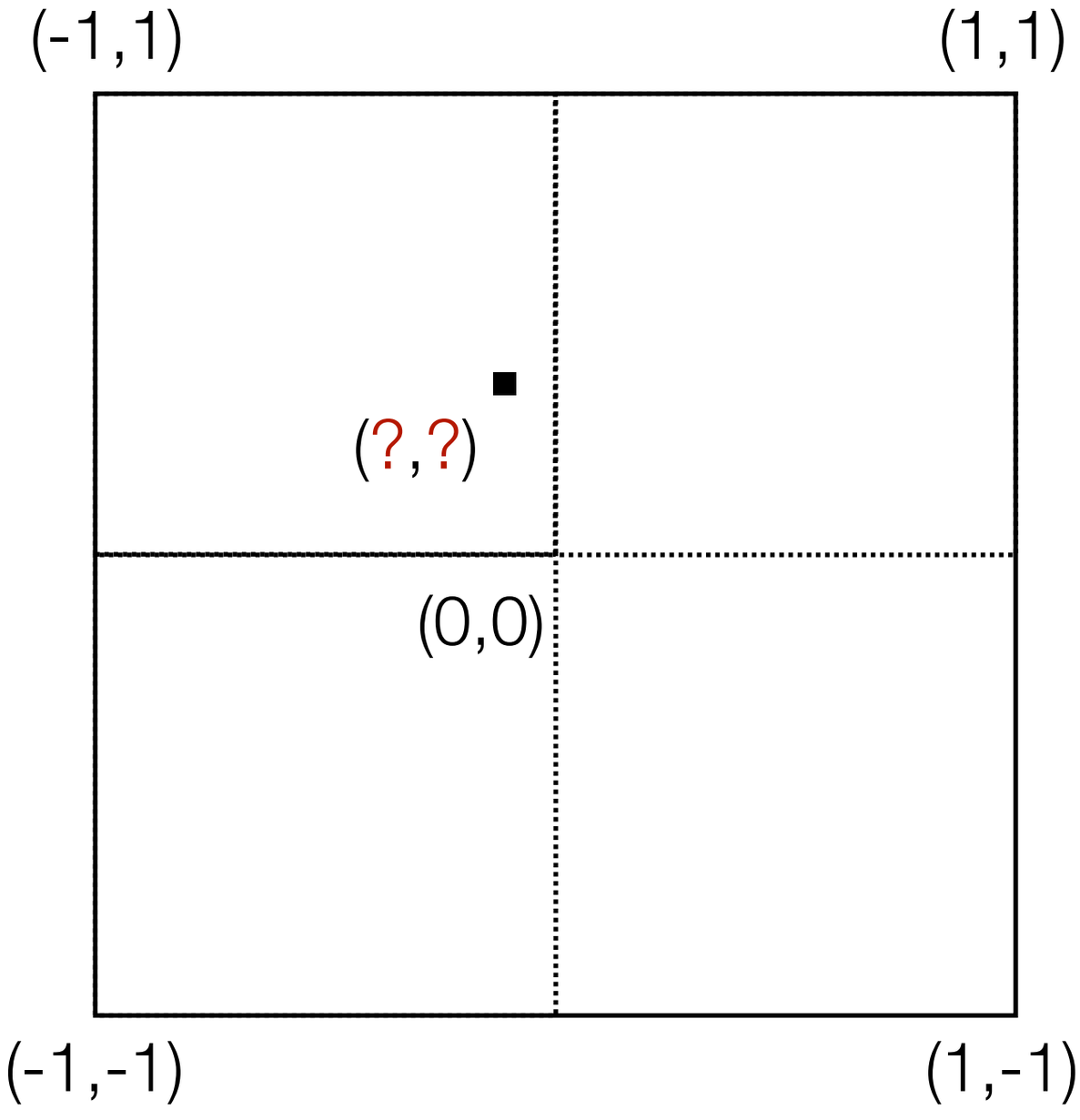
- We can add mouse event listeners to handle input.
  - http://jogamp.org/deployment/v2.3.2/javadoc/jogl/javadoc/com/jogamp/newt/event/MouseListener.html
- Adaptors let us only handle the events we care about.
  - http://jogamp.org/deployment/v2.3.2/javadoc/jogl/javadoc/com/jogamp/newt/event/MouseAdapter.html
- See LineDrawing.java

Mouse Events

- When we click on the screen we get the mouse co-ordinates in screen co-ordinates.
- We need to somehow map them back to viewport co-ordinates.

Mouse Events





Event handling

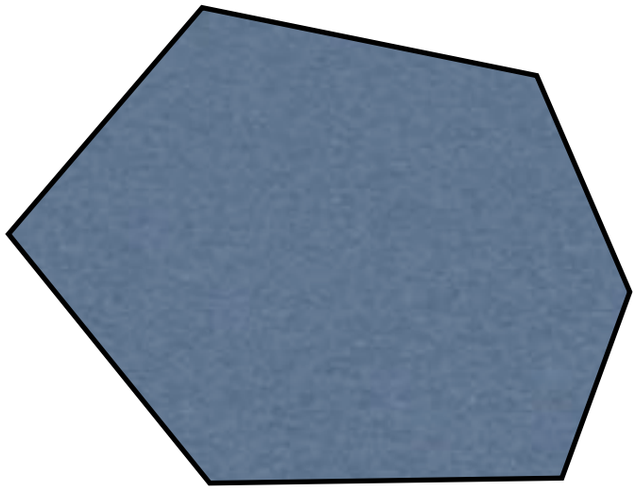
- GL commands should generally only be used within the GLEventListener events
  - Don’t try to store GL objects and use GL commands in mouse listeners.
- In multi-threaded code it is easy to create a mess if you write the same variables in different threads.

Triangles

- We can draw triangles with GL_TRIANGLES
- See Triangle2D.java and TriangleDrawing.java

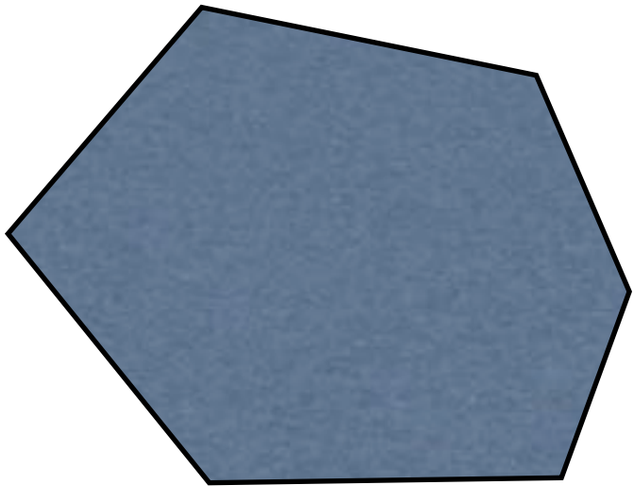
Polygons

- Shapes with an arbitrary number of sides

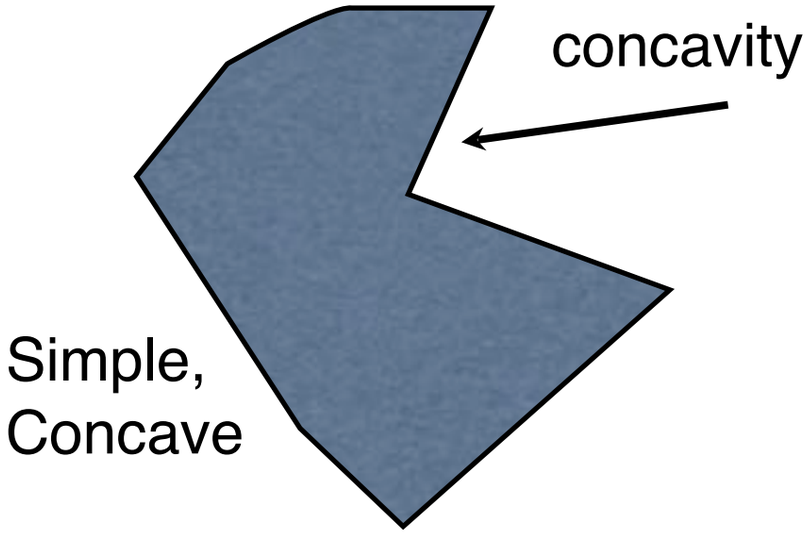


- Whether or not we can easily draw them depends on a few factors

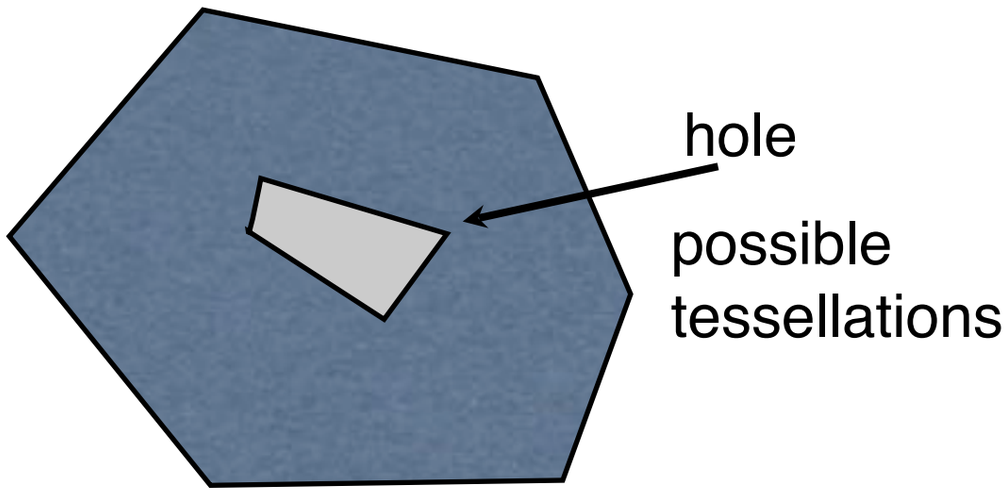
Polygons



Simple, Convex



Simple,
Concave

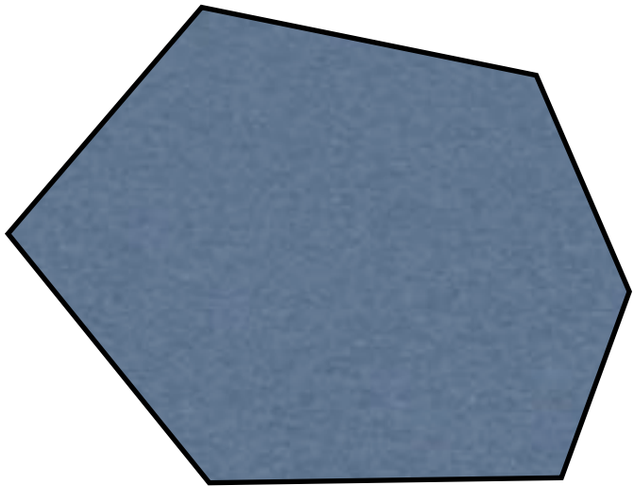


Not simple

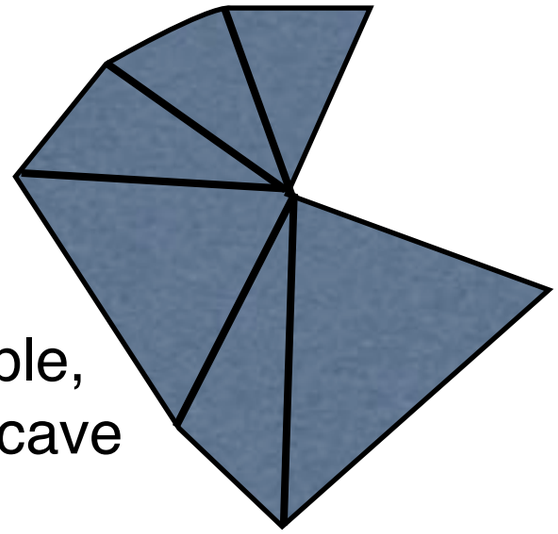
Tessellation

- We draw polygons by splitting them up into simpler shapes (typically triangles)

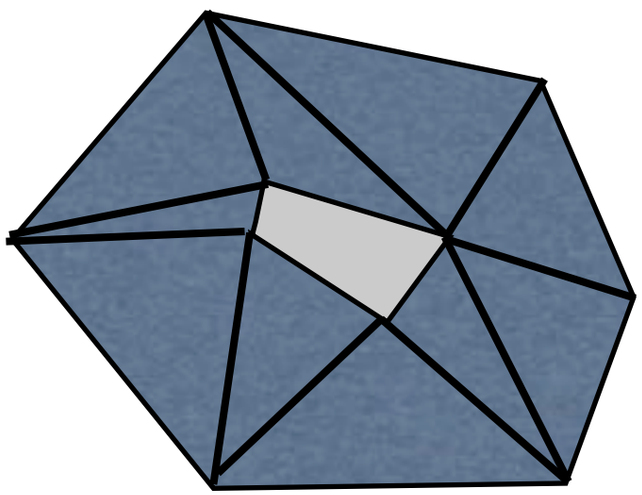
Tessellation



Simple, Convex



Simple,
Concave



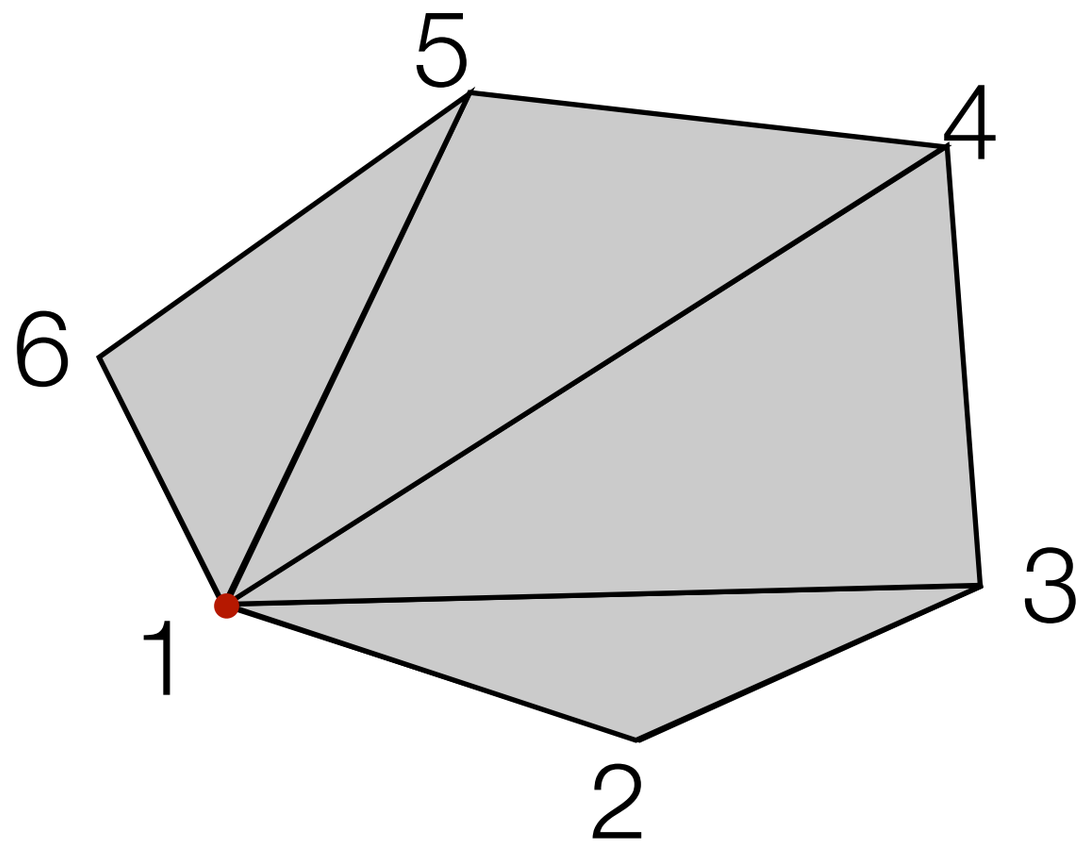
possible
tessellations

Not simple

Triangle Fans

- One simple method is to use a triangle fan.
- Start with any vertex of the polygon and move clockwise or counter-clockwise around it.
- The first three points form a triangle. Any new points after that form a triangle with the last point and the starting point.

Triangle Fans



Triangle Fans

- Works for all simple convex polygons, and some concave ones
- Can be drawn with GL_TRIANGLE_FAN
- The lab task