# 1. Introduction

COMP3421 Computer Graphics • KC Notes

## 1.1 Physics

* Visible spectrum 400 to 700 nanometres – pure spectral light
  + Non-spectral, e.g. white, purple, pink, brown – mixtures of lights
  + Retina with rods for low-level lighting, cones for brighter light for detail, colour
* Images
  + Output: 2D array of RGB colours, rendered at 30 or 60 frames per second
* GPU has its own graphics memory (VRAM) and a frame buffer
  + Has many cores specialised for parallel work – single instruction multiple data
  + OpenGL: low-level 2D/3D graphics API designed for GPU
  + JOGL: Java wrapper around OpenGL (C library) with NEWT, a windowing toolkit

## 1.2 Drawing points

|  |  |
| --- | --- |
| 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  program starts, gives control to a controller  display handler – called when program wants to display on the screen  draw function (see below) |
|  |  |

* **Event-driven**: main sets up components, registers event handlers
  + Events are then dispatched by the event loop
  + Handlers are called when events occur, e.g. display()
* **Viewport**: rectangle, has own coordinate system, by default centred at (0, 0) with +/- 1 boundaries
* OpenGL not OO, uses ints and lots of ‘global state’

|  |  |
| --- | --- |
| public void draw(GL3 gl, CoordFrame2D frame) {  Point2DBuffer buffer = new Point2DBuffer(1);  buffer.put(0, this);  int[] names = new int[1];  gl.glGenBuffers(1, names, 0);  gl.glBindBuffer(GL.GL\_ARRAY\_BUFFER, names[0]);  gl.glBufferData(GL.GL\_ARRAY\_BUFFER,   2 \* Float.BYTES, buffer.getBuffer(),  GL.GL\_STATIC\_DRAW);  gl.glVertexAttribPointer(Shader.POSITION, 2,  GL.GL\_FLOAT, false, 0, 0);  Shader.setModelMatrix(gl, frame.getMatrix());  gl.glDrawArrays(GL.GL\_POINTS, 0, 1);  gl.glDeleteBuffers(1, names, 0);  } | create a buffer of size 1  transfer buffer to GPU  draw point  free buffer |
|  |  |

Point2DBuffer buffer = new Point2DBuffer(1);  
buffer.put(0, this);

* Creates and puts point into buffer
* Buffer in main memory containing point coordinates
* ‘Pinned’ in memory – not a Java array as garbage collector may move things, OpenGL expects array to not move

int[] names = new int[1];  
gl.glGenBuffers(1, names, 0); // number of names, array to put names  
gl.glBindBuffer(GL.GL\_ARRAY\_BUFFER, names[0]); // target, name

* Buffers need names, so ask OpenGL to generate a buffer name
* Use the name – OpenGL has global state, so need to set the buffer and all future buffer operations will be performed on this buffer

gl.glBufferData(GL.GL\_ARRAY\_BUFFER, 2 \* Float.BYTES,   
 buffer.getBuffer(), GL.GL\_STATIC\_DRAW); // target, size, data, usage

* Allocate and transfer buffer in graphics memory
* Size must be specified in bytes (4 bytes in a float)
* Buffer usage: GL\_STATIC\_DRAW – data modified once, used many times  
  GL\_DYNAMIC\_DRAW – data modified repeatedly and used repeatedly

gl.glVertexAttribPointer(Shader.POSITION, 2, GL.GL\_FLOAT, false, 0, 0);  
// index, size, type, normalized, stride, offset

* Tell OpenGL that buffer contains vertex positions
  + **Vertex**: the points (corners) that define a geometric shape (with order)
  + Can attach attributes to them, e.g. position

gl.glDrawArrays(GL.GL\_POINTS, 0, 1);  
// mode (primitive), first (starting vertex), count (number of vertices)

* Draw buffer to screen (based on current buffer and vertex positions

gl.glDeleteBuffers(1, names, 0);  
// n, buffers, buffers\_offset

* Delete buffers

## 1.3 Drawing lines

* Line made up of two points

|  |  |
| --- | --- |
| public void draw(GL3 gl, CoordFrame2D frame) {  Point2DBuffer buffer = new Point2DBuffer(2);  buffer.put(0, start);  buffer.put(1, end);  int[] names = new int[1];  gl.glGenBuffers(1, names, 0);  gl.glBindBuffer(GL.GL\_ARRAY\_BUFFER, names[0]);  gl.glBufferData(GL.GL\_ARRAY\_BUFFER,   4 \* Float.BYTES, buffer.getBuffer(),  GL.GL\_STATIC\_DRAW);  gl.glVertexAttribPointer(Shader.POSITION, 2,  GL.GL\_FLOAT, false, 0, 0);  Shader.setModelMatrix(gl, frame.getMatrix());  gl.glDrawArrays(GL.GL\_LINES, 0, 2);  gl.glDeleteBuffers(1, names, 0);  }  @Override  public void display(GL3 gl) {  *super*.display(gl);  Point2D point1 = *new* Point2D(0f, 0f);  Point2D point2 = *new* Point2D(0f, 0f);  Line2D line = new Line2D(point1, point2);  line.draw(gl);  } | Class has two points start and end  This can work with GL\_POINTS as well, and will place the two points |
|  |  |

## 1.4 Drawing line strips

* Java has buffers (java.nio.Buffer), e.g. FloatBuffer
* LineDrawing: mouse events, **transform screen to viewport** co-ordinates

|  |  |
| --- | --- |
| public Point2DBuffer(List<Point2D> points) {  this.capacity = points.size();  floatBuffer = GLBuffers  .newDirectFloatBuffer(capacity \* 2);  for (int i = 0; i < points.size(); i) {  floatBuffer.put(i\*2, points.get(i).getX());  floatBuffer.put(i\*2+1, points.get(i).getY());  }  } | Place points into buffer |
| public void draw(GL3 gl, CoordFrame2D frame) {  Point2DBuffer buffer = new Point2DBuffer(points);  ...  gl.glBufferData(GL.GL\_ARRAY\_BUFFER,   points.size \* 2 \* Float.BYTES,   buffer.getBuffer(), GL.GL\_STATIC\_DRAW);  gl.glVertexAttribPointer(Shader.POSITION, 2,  GL.GL\_FLOAT, false, 0, 0);  Shader.setModelMatrix(gl, frame.getMatrix());  gl.glDrawArrays(GL.GL\_LINE\_STRIP, 0,   points.size());  ...  } | Points from class  This can work with GL\_POINTS as well, and will place the two points |
|  |  |

## 1.6 Polygons

* Drawn by tessellation – simple triangles
* **Triangle fan**: start with any vertex of the polygon, move clockwise or counter-clockwise around it
  + First three points form a triangle
  + New points form triangles with the last point and the starting point: 123, 134, 145, …
  + GL\_TRIANGLE\_FAN

