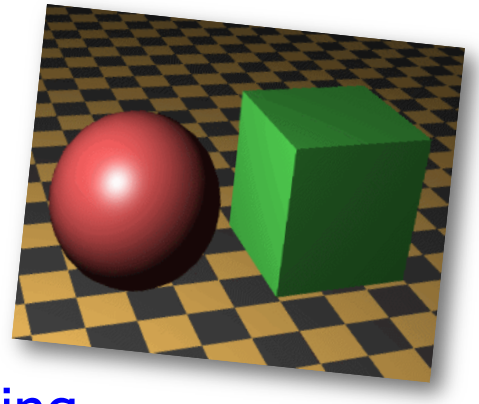
MANCHESTER 1824

COMP27112

Computer Graphics and Image Processing



2: Introducing image synthesis

Toby.Howard@manchester.ac.uk

1

MANCHESTER 1824

Introduction

1. Some orientation – how did we get here?
2. Graphics system architecture
3. Fixed and programmable pipelines
4. Overview of OpenGL / GLU / GLUT
5. When does CGI become "real" ?

- These notes should be read with the OpenGL programming manual – that's where all the detailed programming information is

2

MANCHESTER 1824

Whirlwind (1951)

https://sites.google.com/site/btxprojectcomputergraphics/history-timeline

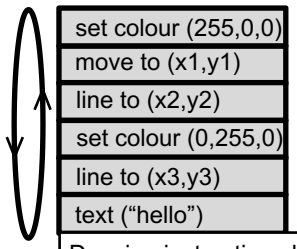
MANCHESTER 1824

SAGE (1955)

4

MANCHESTER 1824

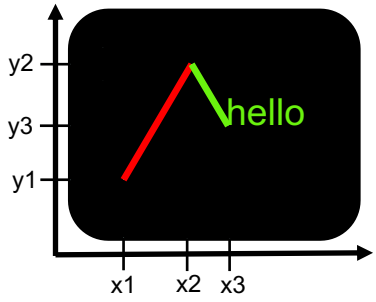
Vector CRT display



Display electronics

Electron beam draws lines and text as instructed. But draws nothing else. Image fades out almost immediately, so must be quickly re-drawn.

Drawing instructions held in memory, repeatedly sent to display, to keep the image visible.

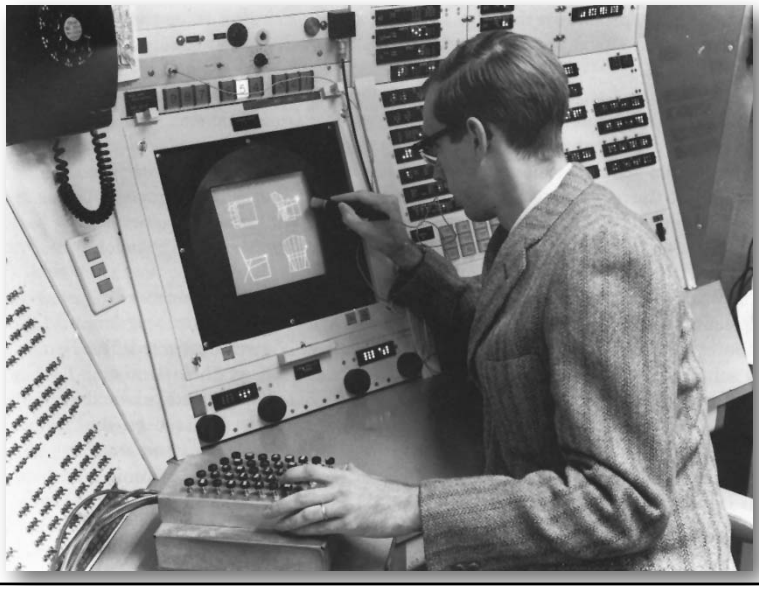


See for example https://www.youtube.com/watch?v=57wj8diYpgY

5

MANCHESTER 1824

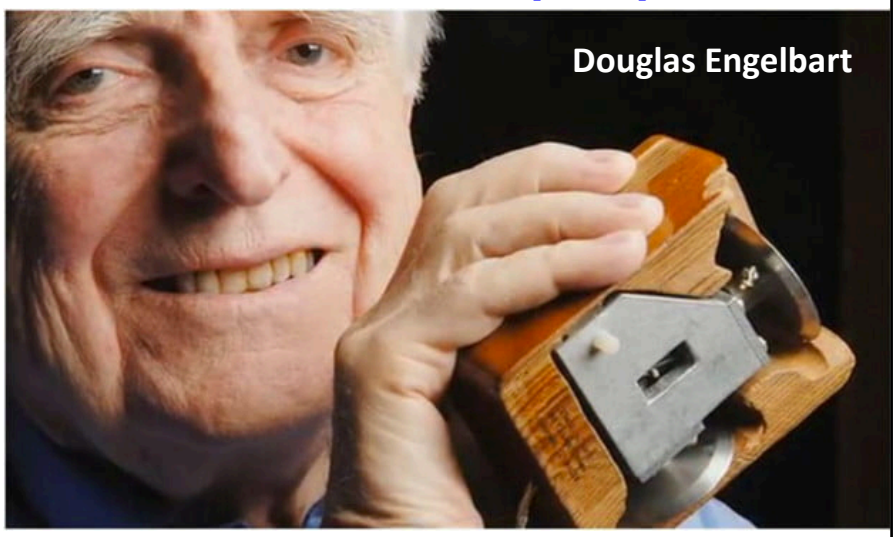
Ivan Sutherland's Sketchpad (1963)



6

MANCHESTER 1824

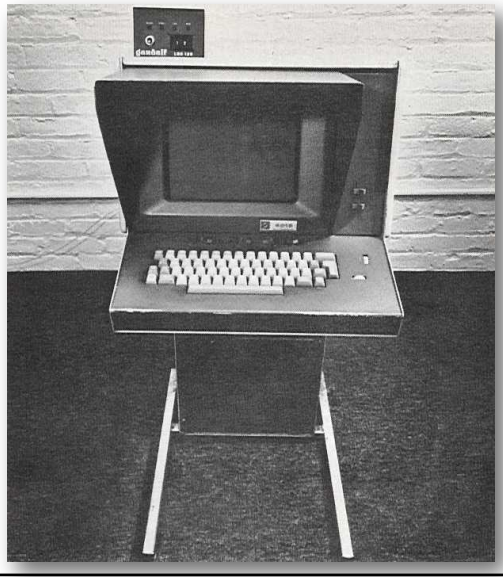
The first mouse (1964)



7

MANCHESTER 1824

Tektronix 4010 vector display (1968)

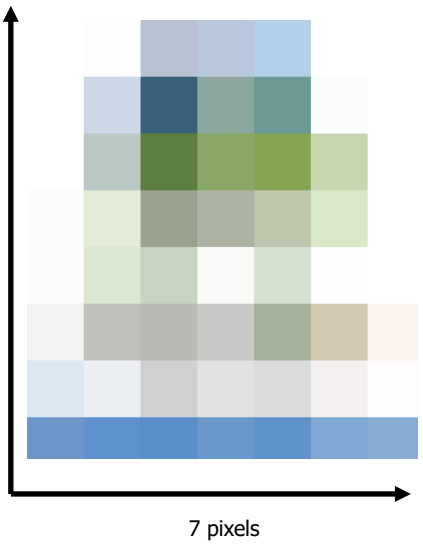


8

MANCHESTER 1824

Raster graphics displays

- Raster graphics uses a 2D array of pixels
- So images must be sampled
- The more samples, the better the fidelity
- But always an approximation

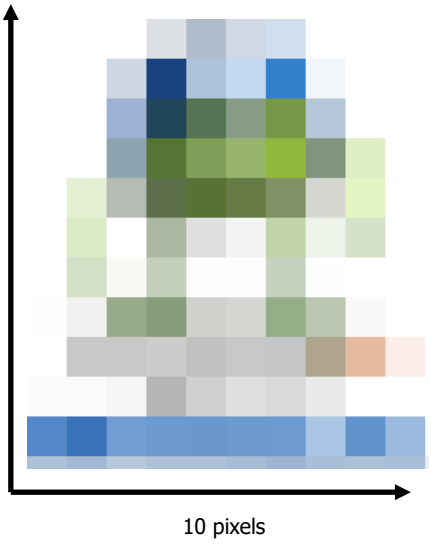


9

MANCHESTER 1824

Raster graphics displays

- Raster graphics uses a 2D array of pixels (screens) or dots (printers)
- So images must be sampled
- The more samples, the better the fidelity
- But always an approximation



10

MANCHESTER 1824

Raster graphics displays

- Raster graphics uses a 2D array of pixels (screens) or dots (printers)
- So images must be sampled
- The more samples, the better the fidelity
- But always an approximation

11

MANCHESTER 1824

Raster graphics displays

- Raster graphics uses a 2D array of pixels (screens) or dots (printers)
- So images must be sampled
- The more samples, the better the fidelity
- But always an approximation

12

MANCHESTER 1824

Raster graphics displays

- Raster graphics uses a 2D array of pixels (screens) or dots (printers)
- So images must be sampled
- The more samples, the better the fidelity
- But always an approximation

39 pixels

13

MANCHESTER 1824

Raster graphics displays

- Raster graphics uses a 2D array of pixels (screens) or dots (printers)
- So images must be sampled
- The more samples, the better the fidelity
- **But always an approximation**

313 pixels

14

MANCHESTER 1824

in
computer graphics
everything
is an approximation

MANCHESTER 1824

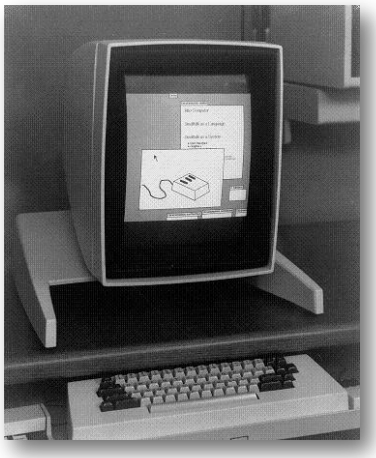
some
approximations
are better
than others

MANCHESTER 1824

Henri Gouraud invents shading (1970)

MANCHESTER 1824

Graphical user-interface (Xerox, 1973)

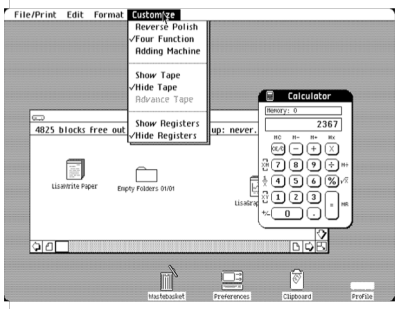


Xerox Alto – 606 x 808 pixels,
6MHz CPU, 96-256kB RAM,
$12,000-$40,000

18

MANCHESTER 1824

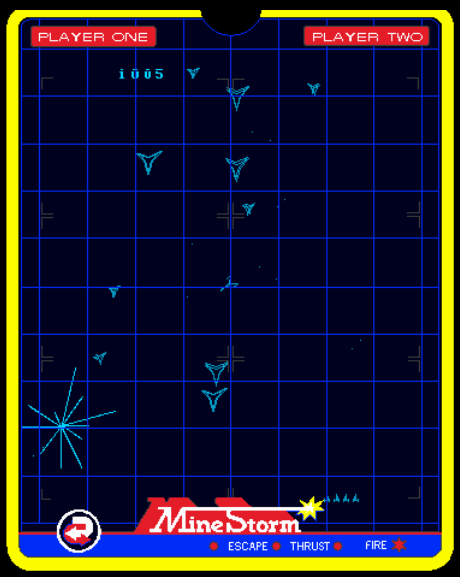
Graphical user-interface (Apple, 1983)



Apple Lisa: 720 x 360 pixels,
5MHz CPU, 1-2MB RAM, $9,995

19

Vectrex vector games console, 1983



MANCHESTER 1824

OpenGL

- Open Graphics Library
- platform/language-independent
- origins in GL, by Silicon Graphics, Inc.
- OpenGL first released 1992
- Today, open standard for portable graphics programming
- www.opengl.org

21

MANCHESTER 1824

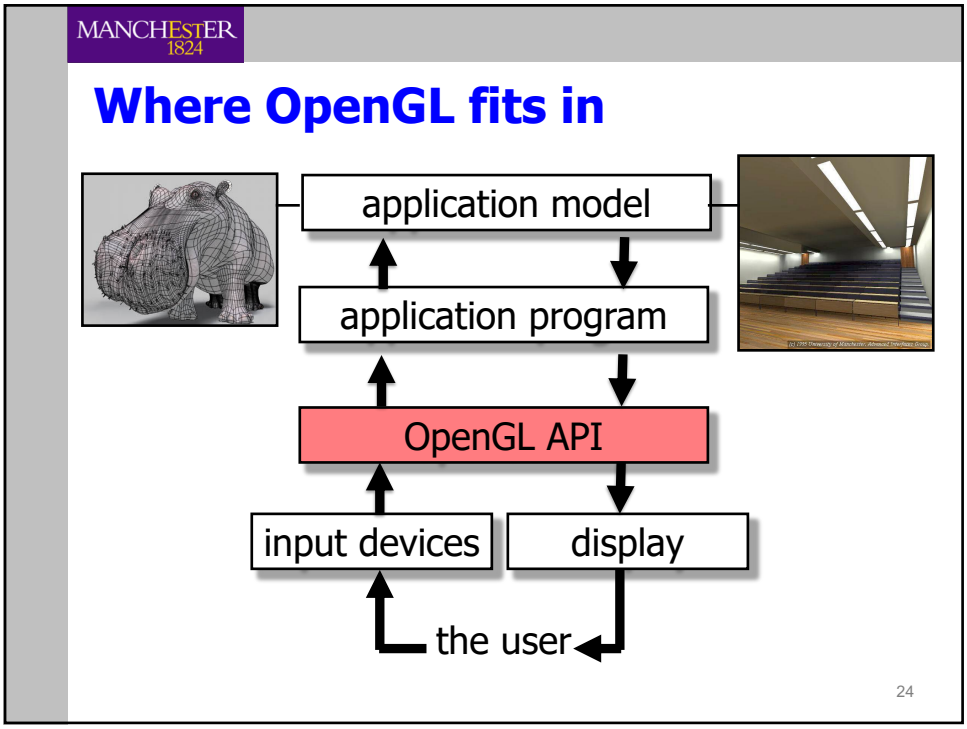
OpenGL vs. DirectX

- OpenGL
  - Open standard, run by the **Khronos Group**
  - Cross-architecture (Win/Mac/Linux…)
- DirectX
  - Owned by Microsoft

- Functionally, the two systems are roughly the same
- For an in-depth comparison, see http://en.wikipedia.org/wiki/Comparison_of_OpenGL_and_Direct3D

22


BOARD OF PROMOTERS
KHRONOS GROUP
Over 100 members worldwide
any company is welcome to join
AMD
ARM
Apple
SONY
SAMSUNG
EPIC GAMES
intel
VIVANTE
NOKIA
NVIDIA
Imagination
QUALCOMM


MANCHESTER 1824
Where OpenGL fits in
application model
application program
OpenGL API
input devices
display
the user
24

MANCHESTER 1824

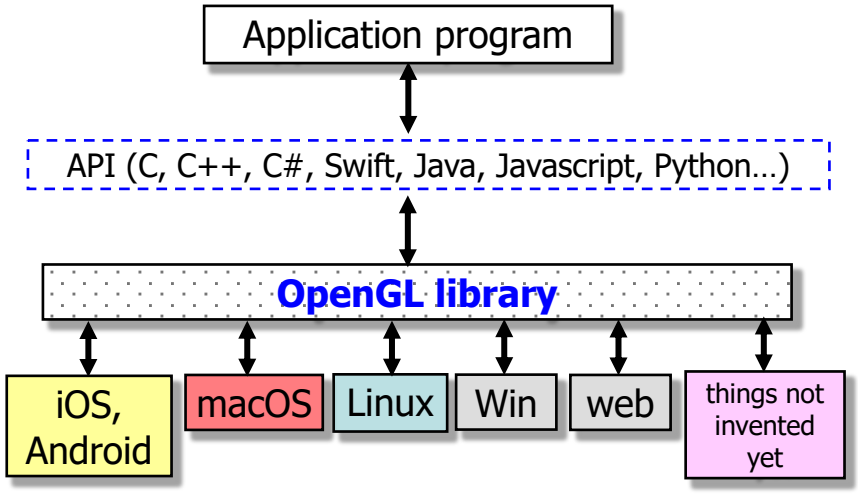
The OpenGL API

- OpenGL is a **specification** of an Application Programmer's Interface (API), so language in
- A set of functions for doing 3D computer graphics
- used in mobile devices, industry, engineering, CAD, CGI, FX, research, games, education – everywhere

25

MANCHESTER 1824

Portability



26

MANCHESTER 1824

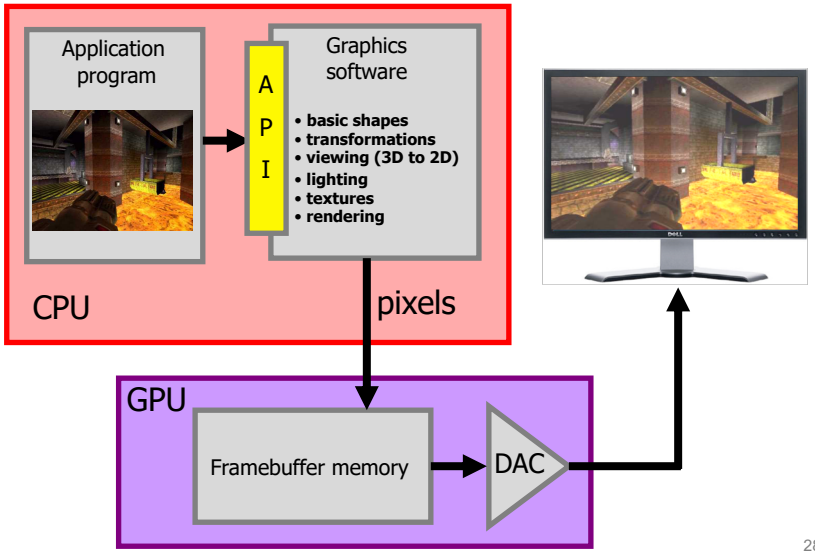
OpenGL evolution

- OpenGL functionality has evolved over time:
- OpenGL v1, v2: fixed pipeline – fixed functionality
- OpenGL v3+: programmable pipeline – extensible functionality: programmers write micro-programs called **shaders**
- OpenGL 4.6 (2017)

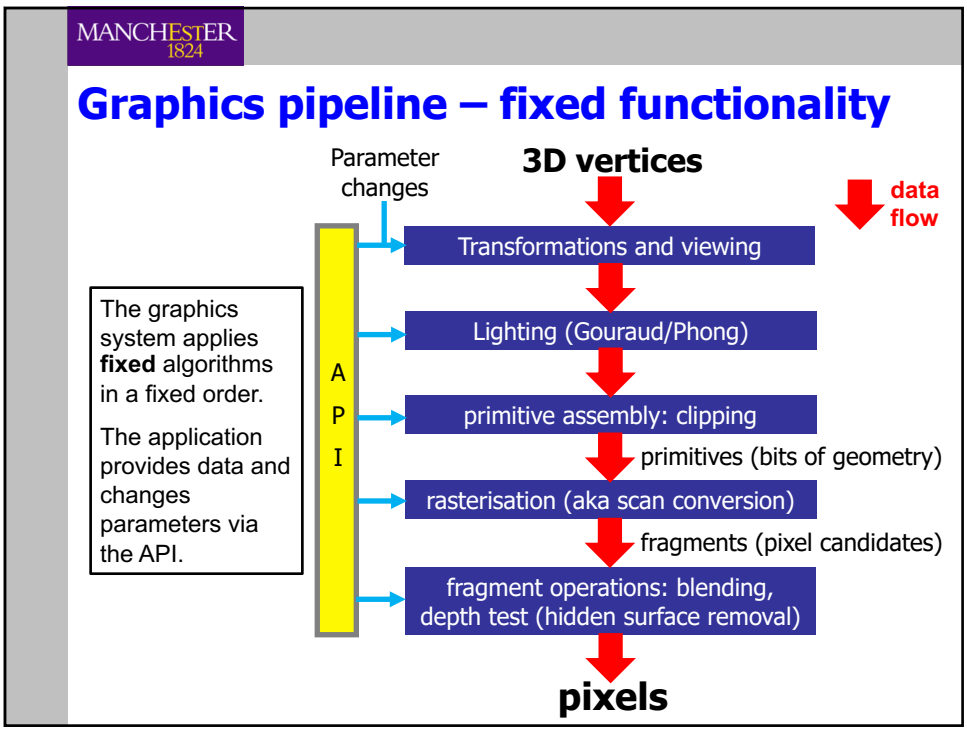
27

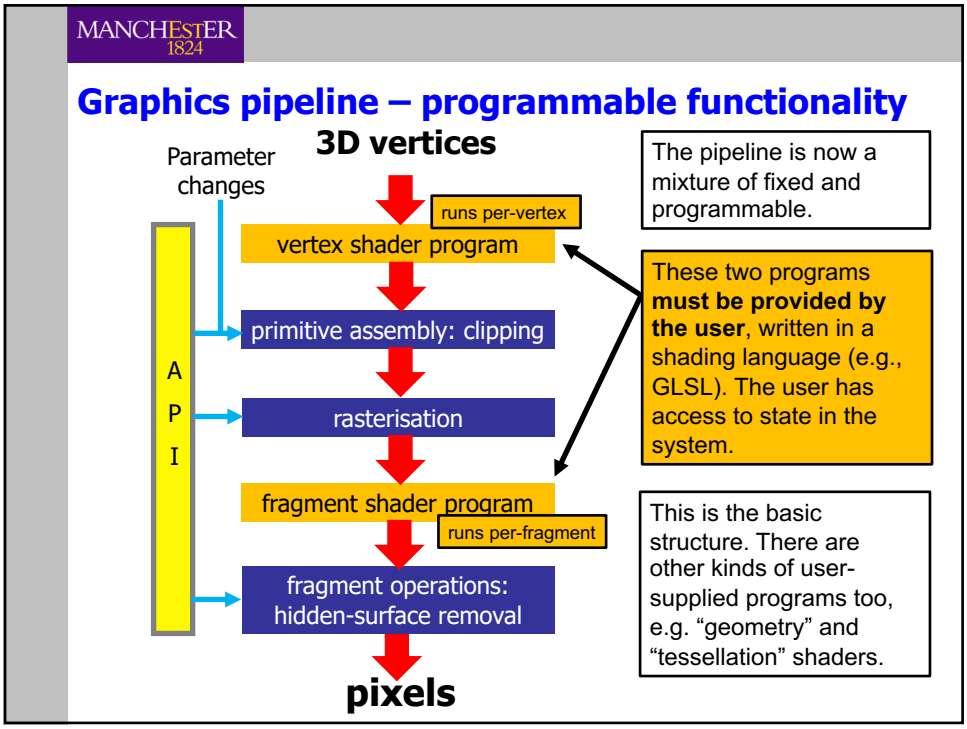
MANCHESTER 1824

Basic graphics system architecture



28


MANCHESTER 1824
Graphics pipeline – fixed functionality
Parameter changes
3D vertices
data flow
Transformations and viewing
Lighting (Gouraud/Phong)
primitive assembly: clipping
primitives (bits of geometry)
rasterisation (aka scan conversion)
fragments (pixel candidates)
fragment operations: blending, depth test (hidden surface removal)
pixels
A P I
The graphics system applies **fixed** algorithms in a fixed order.
The application provides data and changes parameters via the API.


MANCHESTER 1824
Graphics pipeline – programmable functionality
Parameter changes
3D vertices
runs per-vertex
vertex shader program
primitive assembly: clipping
rasterisation
fragment shader program
runs per-fragment
fragment operations: hidden-surface removal
pixels
A P I
The pipeline is now a mixture of fixed and programmable.
These two programs **must be provided by the user**, written in a shading language (e.g., GLSL). The user has access to state in the system.
This is the basic structure. There are other kinds of user-supplied programs too, e.g. “geometry” and “tessellation” shaders.

MANCHESTER 1824

An example GLSL vertex shader

This is the incoming vertex to be processed next, part of the graphics system's state. The shader can manipulate this however it likes.

This is for context only. It is not examinable

```
attribute vec4 gl_Vertex;
void main() {
gl_Position = gl_ProjectionMatrix * gl_ModelViewMatrix * gl_Vertex;
}
```

This is the result of the program, and is the vertex that now proceeds down the pipeline.

These matrices are part of the graphics system's state and are set by the application via the API.

31

MANCHESTER 1824

An example GLSL fragment shader

This user-defined variable is part of the graphics system's state and can be set via the API.

This is for context only. It is not examinable

```
attribute vec4 myColour;
void main() {
gl_FragColor = myColour;
}
```

This is current fragment's colour.

32

MANCHESTER 1824

Fixed versus programmable

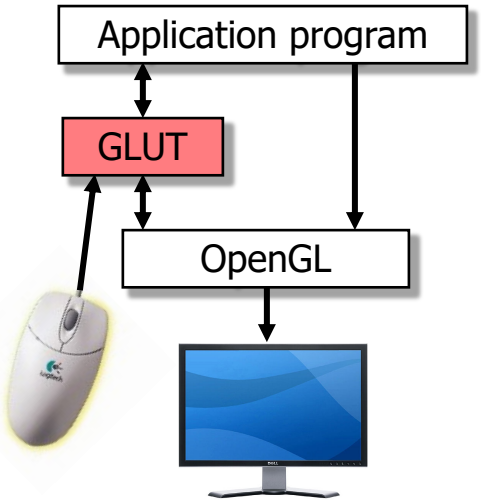
- The fixed pipeline
  - con: new algorithms and techniques can't be added
  - con: it's deprecated
  - pro: it's simple to use and fine for many purposes
  - pro: for the beginner it's easy to get started in a short time
- The programmable pipeline
  - pro: provides maximum flexibility
  - pro: it's the state-of-the-art, cutting edge
  - con: for the beginner there is significant start-up cost
- In COMP27112 we use the fixed pipeline

33

MANCHESTER 1824

OpenGL and interaction

- OpenGL only generates pixels
- it doesn't handle interaction devices
- for interaction, we use the GLUT library*



* there are other GUI libraries

34

MANCHESTER 1824

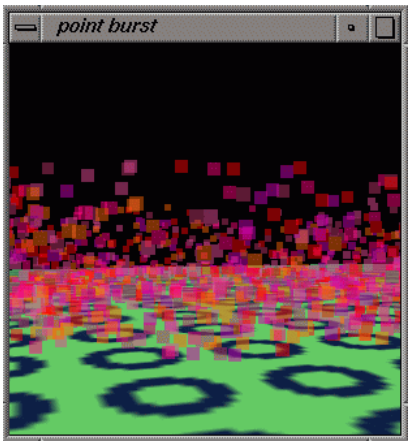
OpenGL graphics

- The main features of OpenGL
  - 3D graphics (points, lines, polygons…)
  - coordinate transformations
  - a camera for viewing
  - hidden surface removal
  - lighting and shading
  - texturing
  - pixel (image) operations
- We'll take a whirlwind tour

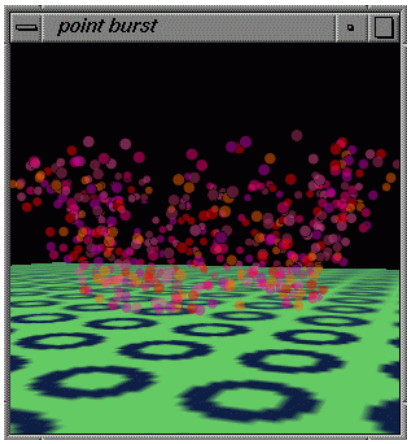
35

MANCHESTER 1824

Points



Regular points



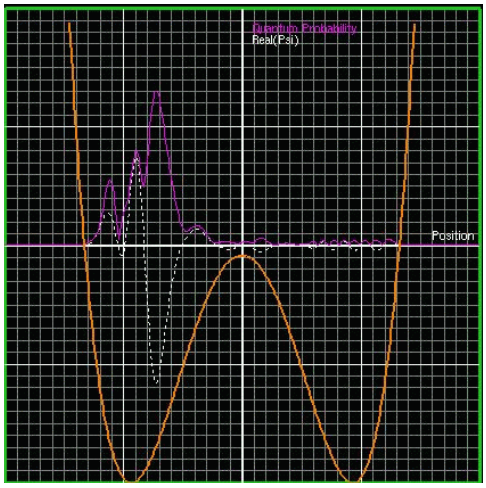
Anti-aliased points

36

MANCHESTER 1824

Lines

- Lines, like all graphical primitives, have two kinds of property:
  - **geometry** (shape)
  - **attributes** (appearance)

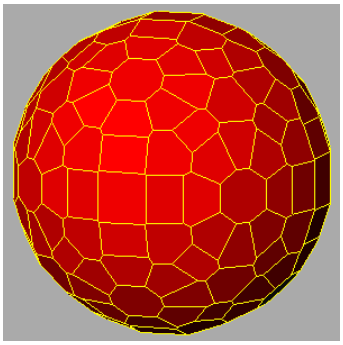


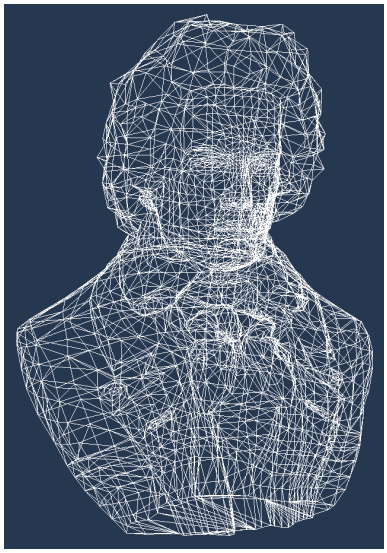
37

MANCHESTER 1824

Polygons

- Triangles
- Quadrilaterals
- Convex polygons



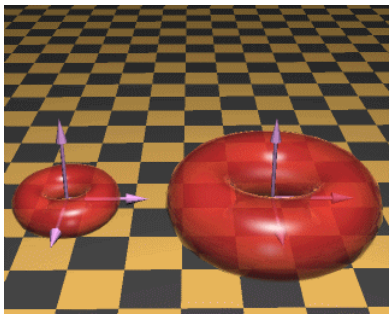


38

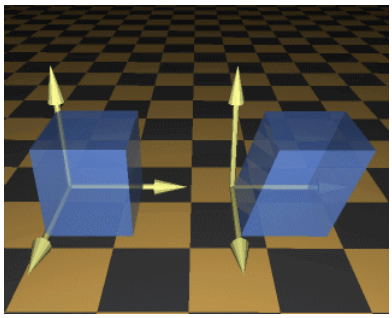
MANCHESTER 1824

Transformations

- OpenGL expresses coordinate transformations as 4x4 homogeneous matrices



scaling

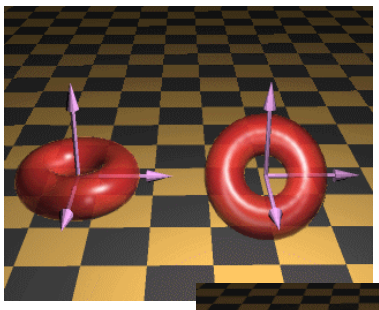


shearing

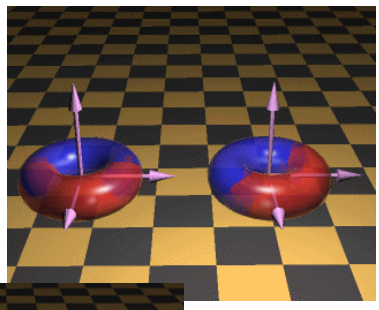
39

MANCHESTER 1824

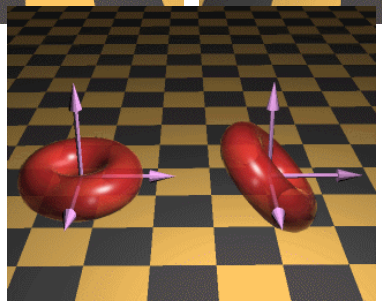
Transformations



rotation about x



rotation about y



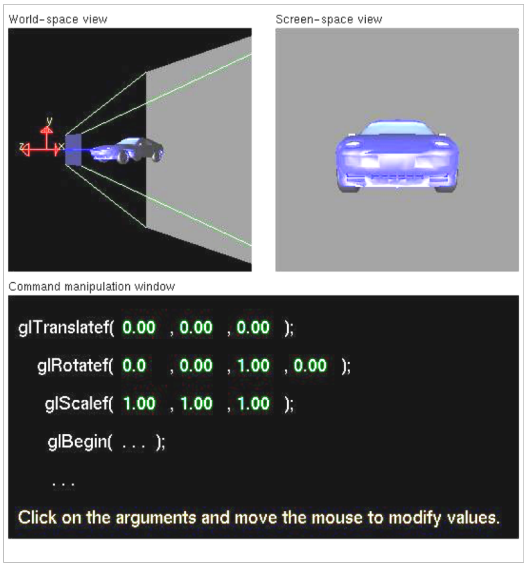
rotation about z

40

MANCHESTER 1824

Transformations

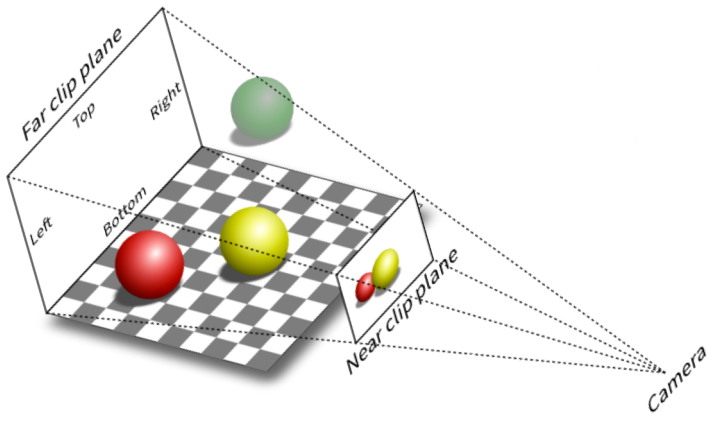
- Transformations can be combined and nested
- So complex objects to be assembled from simpler parts
- You explore this in Lab 2



MANCHESTER 1824

The camera model

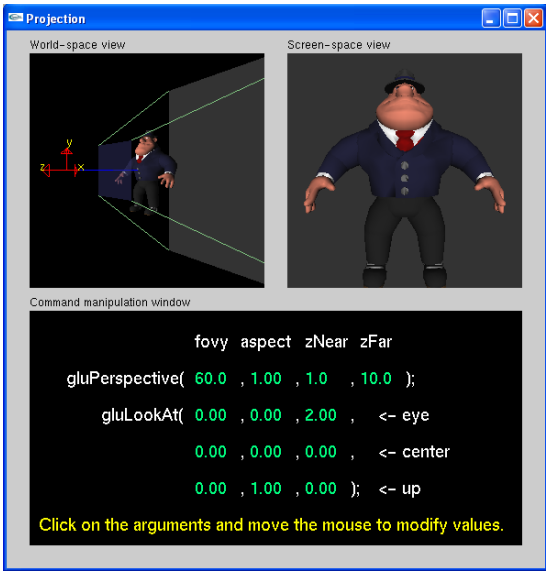
- 2D view of 3D scene. Perspective projection / parallel projection



42

MANCHESTER 1824

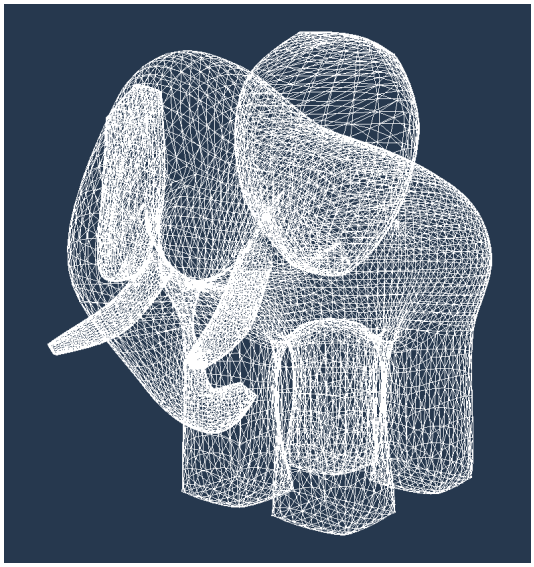
The camera model



43

MANCHESTER 1824

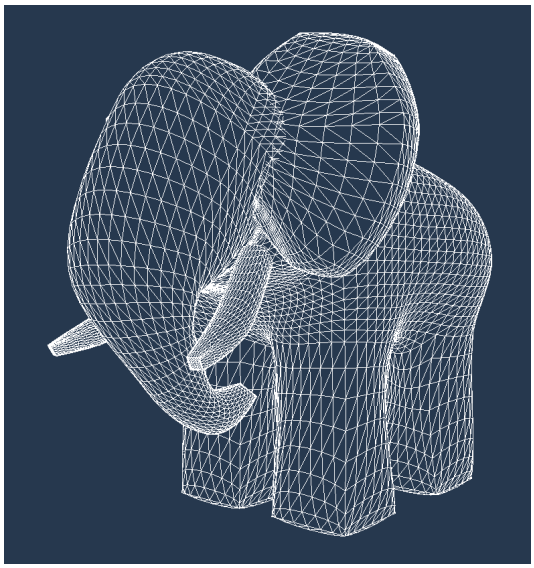
Hidden-surface removal



44

MANCHESTER 1824

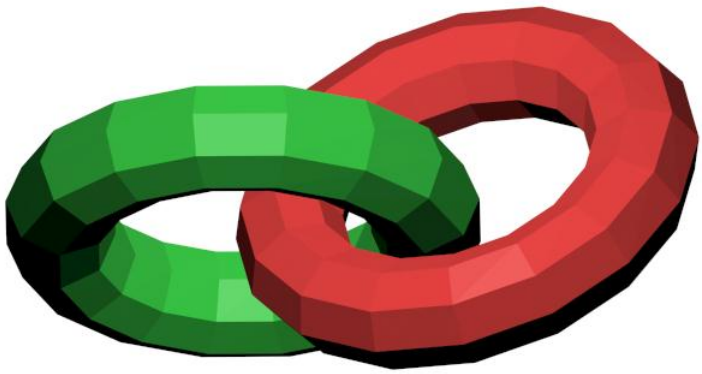
Hidden-surface removal



45

MANCHESTER 1824

Lighting and shading

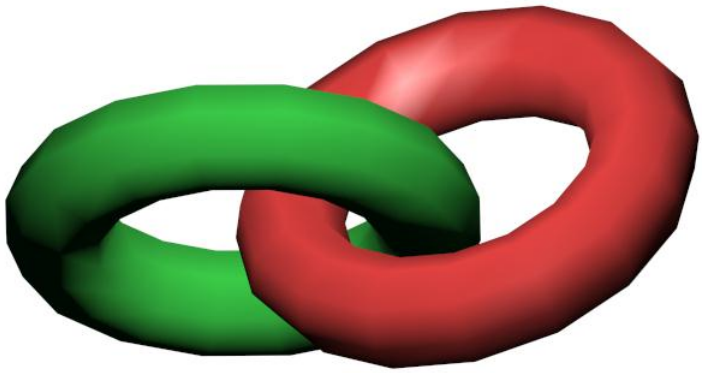


- Local methods: **flat**, Gouraud and Phong

46

MANCHESTER 1824

Lighting and shading

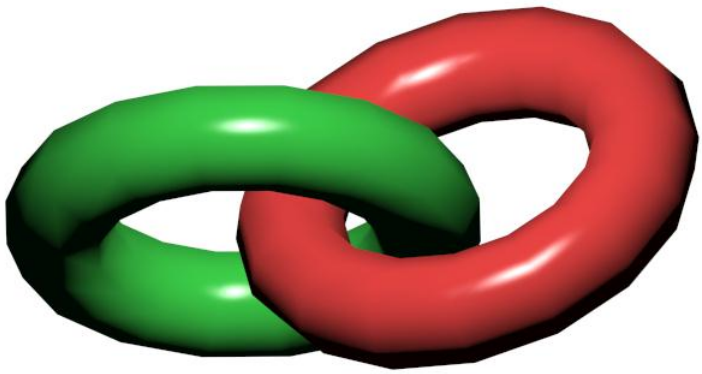


- Local methods: flat, **Gouraud** and Phong

47

MANCHESTER 1824

Lighting and shading



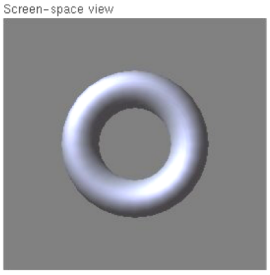
- Local methods: flat, Gouraud and **Phong**

48

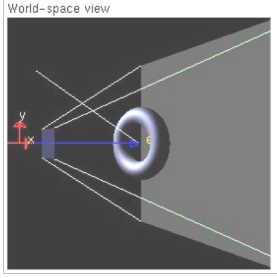
MANCHESTER 1824

Lighting and shading

Screen-space view



World-space view



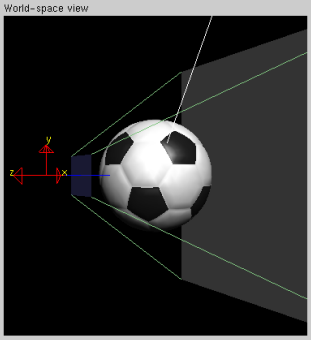
Command manipulation window

```
GLfloat light_pos[ ] = {  -2.00 , 2.00  , 2.00  , 1.00  };
GLfloat light_Ka[ ] = {  0.00  , 0.00  , 0.00  , 1.00  };
GLfloat light_Kd[ ] = {  1.00  , 1.00  , 1.00  , 1.00  };
GLfloat light_Ks[ ] = {  1.00  , 1.00  , 1.00  , 1.00  };

glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_Ka);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Kd);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_Ks);

GLfloat material_Ka[ ] = { 0.11  , 0.06  , 0.11  , 1.00  };
GLfloat material_Kd[ ] = { 0.43  , 0.47  , 0.54  , 1.00  };
GLfloat material_Ks[ ] = { 0.33  , 0.33  , 0.52  , 1.00  };
GLfloat material_Ke[ ] = { 0.00  , 0.00  , 0.00  , 0.00  };
GLfloat material_Se =  10  ;

glMaterialfv(GL_FRONT, GL_AMBIENT, material_Ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, material_Kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, material_Ks);
glMaterialfv(GL_FRONT, GL_EMISSION, material_Ke);
glMaterialfv(GL_FRONT, GL_SHININESS, material_Se);

Click on the arguments and move the mouse to modify values.
```

MANCHESTER 1824

Light position

World-space view



Screen-space view

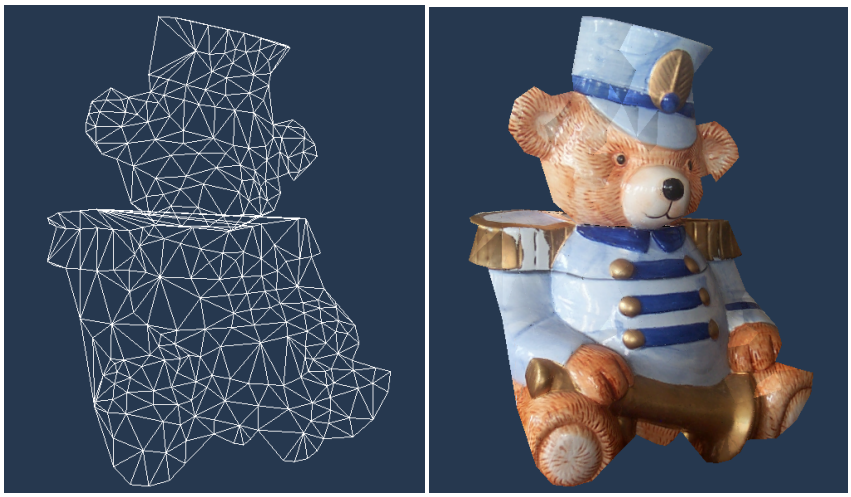
Command manipulation window

```
GLfloat pos[4] = { 1.50  , 1.00  , 1.00  , 0.00  };
      gluLookAt( 0.00  , 0.00  , 2.00  ,   <- eye
                 0.00  , 0.00  , 0.00  ,   <- center
                 0.00  , 1.00  , 0.00  );  <- up
  glLightfv(GL_LIGHT0, GL_POSITION, pos);

Click on the arguments and move the mouse to modify values.
```

50

MANCHESTER 1824

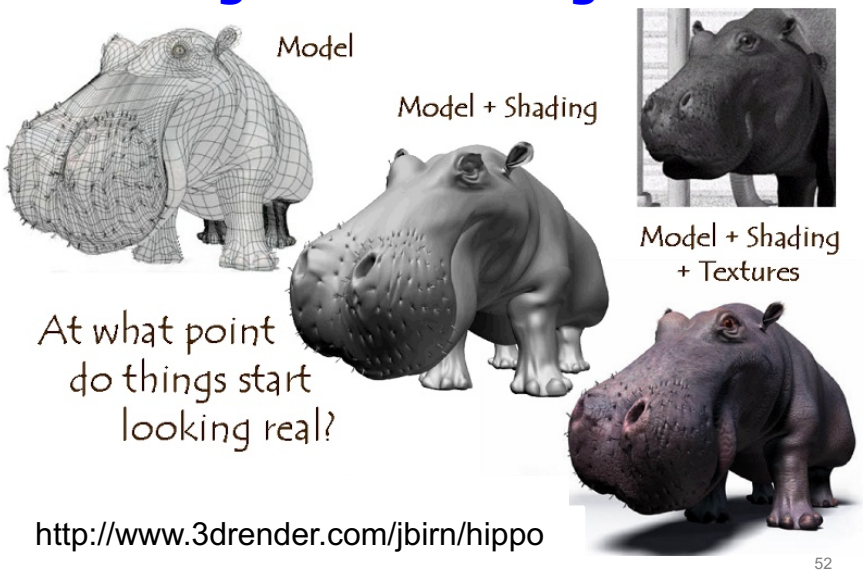
Textures



51

MANCHESTER 1824

Modelling and rendering

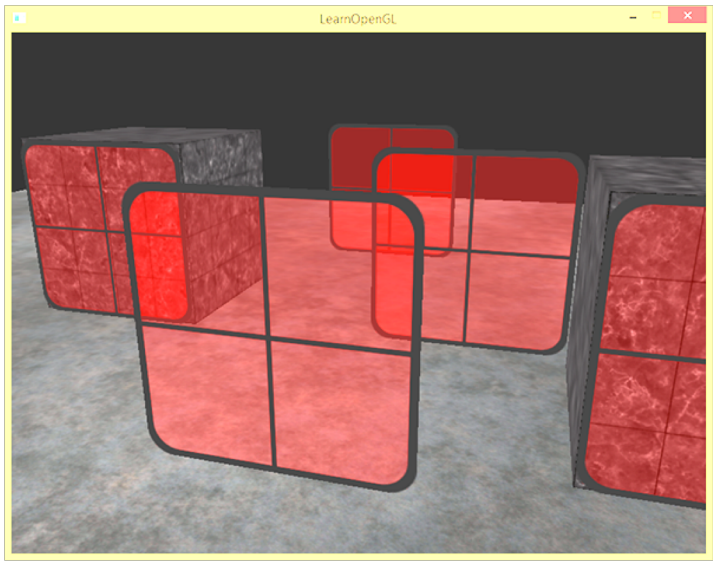


http://www.3drender.com/jbirn/hippo

52

MANCHESTER 1824

Blending/transparency



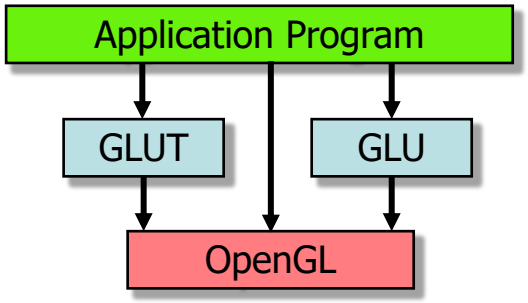
learnopengl.com/Advanced-OpenGL/Blending

53

MANCHESTER 1824

Support libraries

- OpenGL supports low-level rendering only
- two support libraries have been developed
  - GLU and GLUT



MANCHESTER 1824

GLU

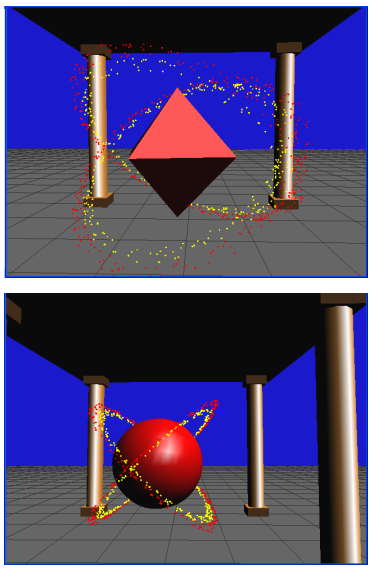
- GL Utility Library (GLU)
- provides functions which "wrap up" lower-level OpenGL graphics
  - Curves, surfaces, cylinder, disc, quadrics...
- Utility functions
  - Viewing
  - Textures
  - Tessellation
- Command reference:
  - http://www.opengl.org/sdk/docs/man/

55

MANCHESTER 1824

GLUT

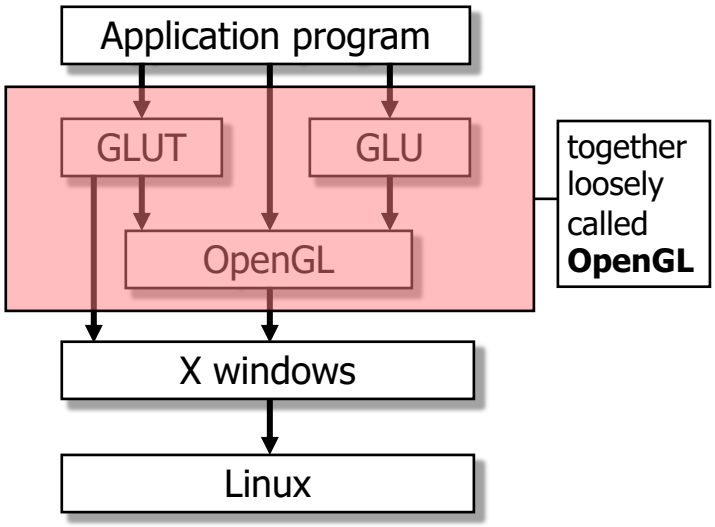
- GL Utility Toolkit (GLUT)
- Interaction
  - interaction (mouse and keyboard)
  - simple menu system
- Primitives
  - Sphere, torus, cone, cube
  - [tetra|octo|dodeca|icosa]hedron
  - teapot
- Command reference:
  - Link to PDF on Course Bb



56

MANCHESTER 1824

OpenGL on Linux



57

MANCHESTER 1824

What do do next

1. OpenGL manual – Tutorial Chapters 1-6
2. Do Coursework 1
3. Start looking at Lab 1