Architecture of a Graphics Pipeline

6 February 2007
CMPT370
Dr. Sean Ho
Trinity Western University

Review last time

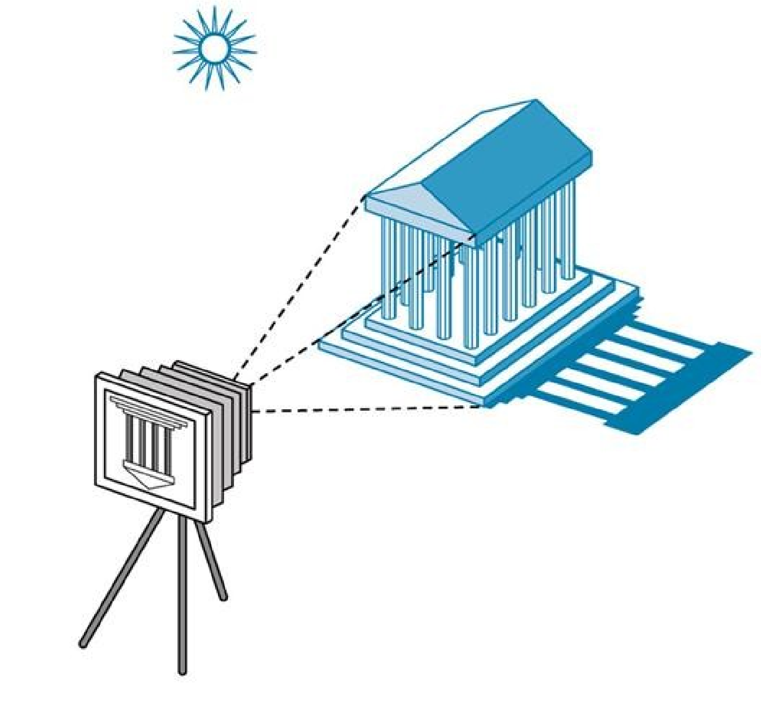
- Visual computing:
  - Computer graphics and image analysis
- Objectives of visual computing
  - Capture and understand reality
  - Emulate and enhance reality
  - Parthenon video
- Image formation
  - Camera model

What's on for today

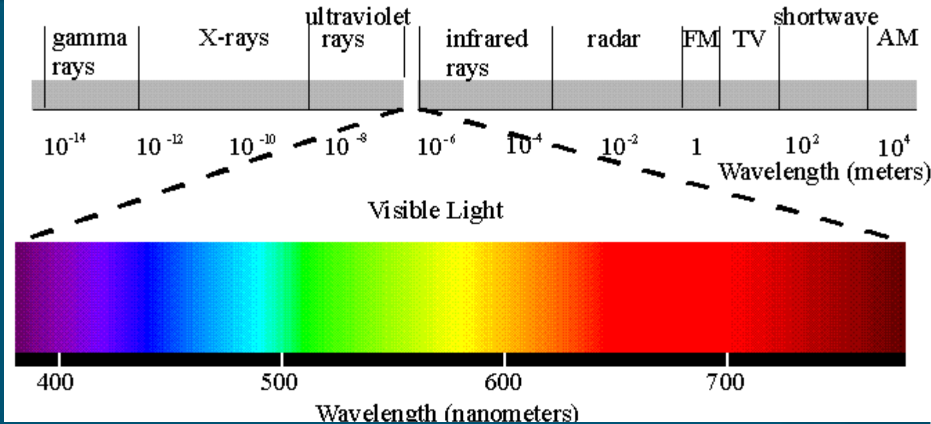
- Light and colour models
- Geometric representation: trimesh
- Off-line rendering: raytracing, radiosity
- Real-time interactive graphics pipeline:
  - Vertex processing
  - Clipping and culling
  - Rasterizing
  - Fragment processing
- Graphics API overview (OpenGL)

Image formation

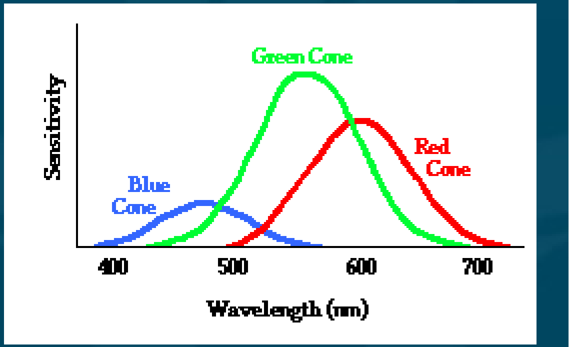
- Components to produce a static image:
  - Objects
    - Geometry (vertices, faces, etc.), material properties: colour, shininess, bumpiness, etc.
  - Light sources
    - Colour spectrum, direction, area, etc.
  - Viewer
    - Camera model: lens, depth of field, etc.



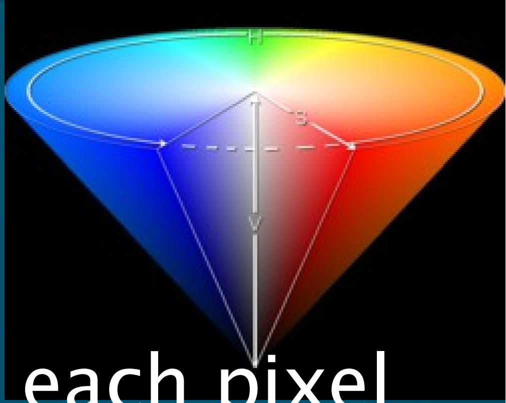
Light



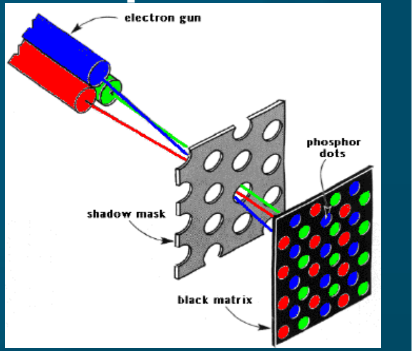
- Visible light is electromagnetic radiation about 350-750nm in wavelength (~400 to 850 THz in frequency)
- Light colour is a frequency distribution of energy
  - Lasers: monochromatic
- But our eyes only have four kinds of sensors:
  - Rods: luminance (shades of grey)
  - R,G,B cones: chrominance (colour)
  - Each sensor has its own frequency response curve



Colour models



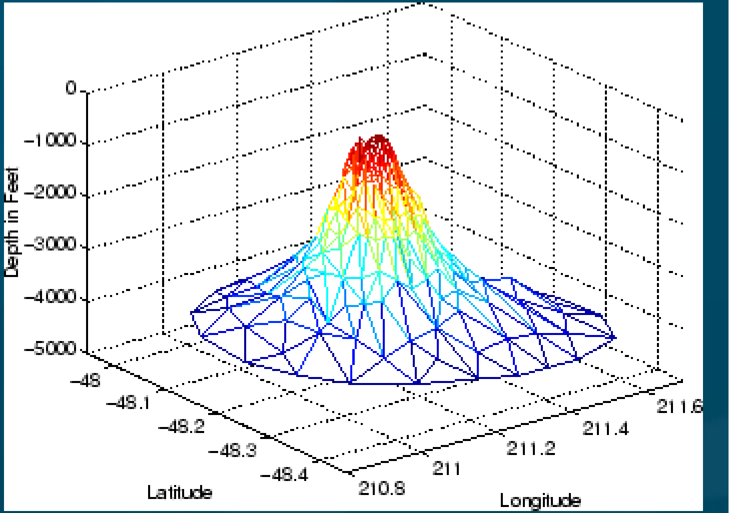
- “True” image: frequency distribution at each pixel



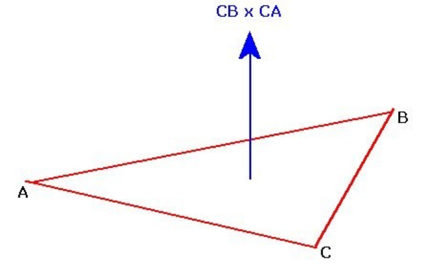
- RGB: matches our cones
  - Additive colour: CRTs use 3 electron guns
  - Must still define chromaticities of R,G,B
- CMYK: subtractive colour: C<->R, M<->G, Y<->B
  - Inks/pigments: newspaper, paint
- HSV: hue, saturation, value
- CIELAB: lightness, a/b chrominance:
  - Absolute colour space: only depends on whitepoint
  - Convert to absolute via profile: AdobeRGB, sRGB

Geometric representation: trimesh

- The most common representation for the geometry of 3D surfaces is a triangle mesh:
  - Vertex list (point cloud): (x,y,z) coordinates
    - {0.2, 0., 2.7}, {0.2, -0.112, 2.7}, {0.112, -0.2, 2.7},
  - Face list: indexes into vertices
    - {12, 13, 14}, {13, 14, 15}, ...



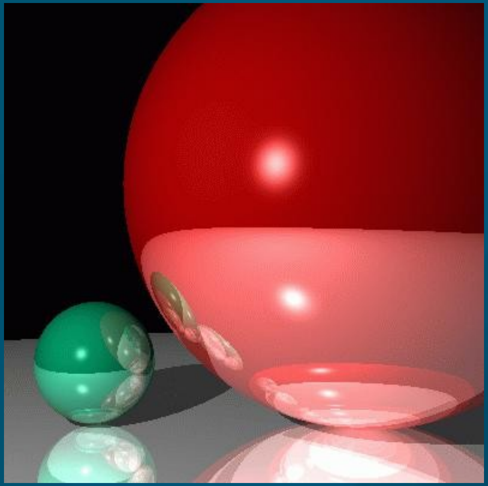
- Can also use other polygons
  - But triangle is a 2D simplex: Always flat
- Faces have normal vectors



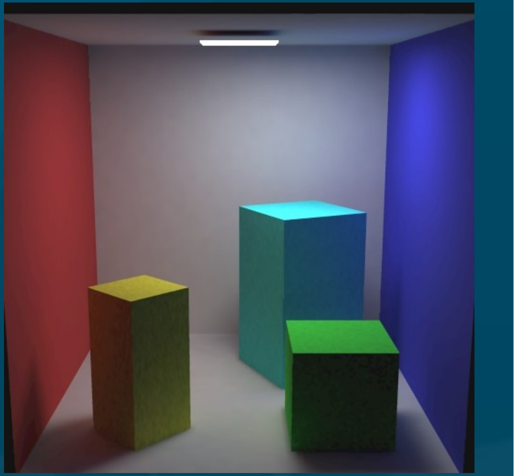
Off-line vs. real-time graphics

- Off-line rendering
  - Render time is not very important
    - Use big parallel render farms
  - Photo-realism is the priority
  - Raytracing, radiosity, other rendering methods
- Real-time (interactive) graphics
  - Perfect photo-realism is not so important
  - Frame rate is the priority: at least 60Hz
  - 3D modelling, CAD, scientific visualization
  - Graphics pipeline in video card or software

Off-line rendering



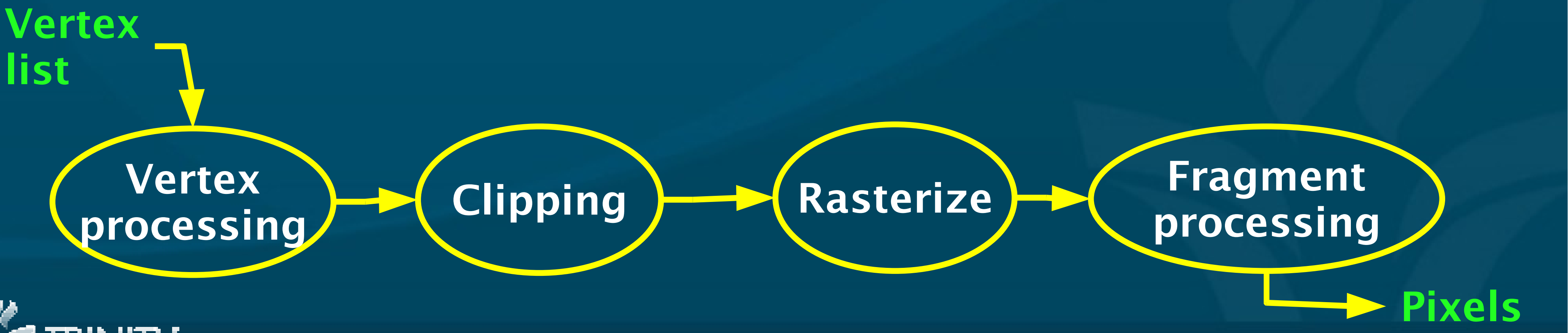
- Raytracing:
  - Cast rays from camera into scene until either absorbed or go to infinity
    - Sky sphere handles infinity
  - Reflections, translucency, refraction
  - Only trace rays that are needed



- Radiosity:
  - Light sources emit energy
  - Follow light energy as it bounces in scene
  - Global illumination: not view-dependent

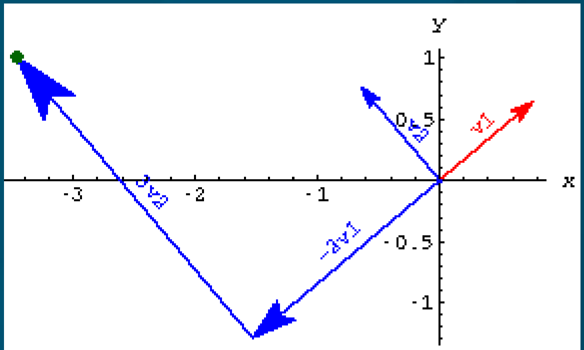
Real-time graphics pipeline

- Process objects one at a time: local lighting
- This is all done in hardware on the graphics card
- Input: scene objects, lighting, camera
  - Most of the data is the vertex list
- Output: pixels stored in the framebuffer
  - Raster graphics



Vertex processing

- Much of the work is in transforming vertices from one coordinate system to another:
  - Object-based coords
  - Camera-based coords
  - Screen-based coords



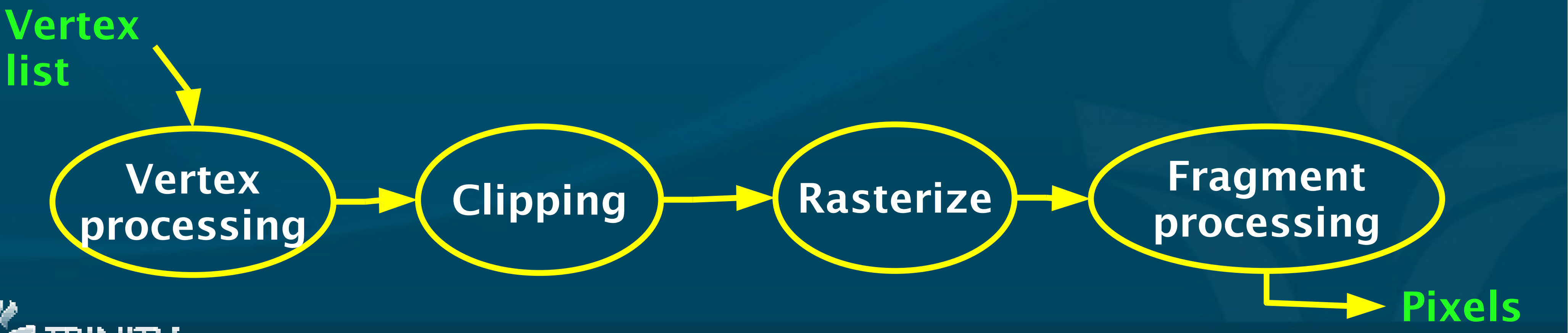
- Each transform is a matrix multiplication
- Vertex processor also computes RGB colour at each vertex

Kinds of coordinate transforms

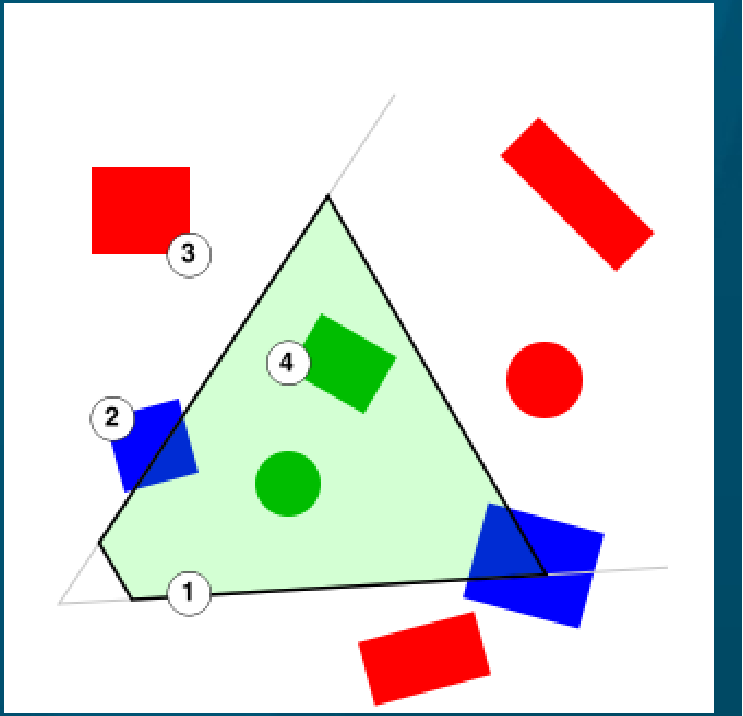
- The transformations done on vertices include:
  - Translation: shift in (x,y,z)
  - Rotation: e.g., 3 Euler angles
  - Scaling: uniform or along 3 axes
  - (Perspective, affine)
- 3D points are projected onto 2D image plane:
  - Perspective projection:
    - Projection lines meet at center of projection
  - Parallel projection:
    - Projection lines are all parallel

Primitive assembly

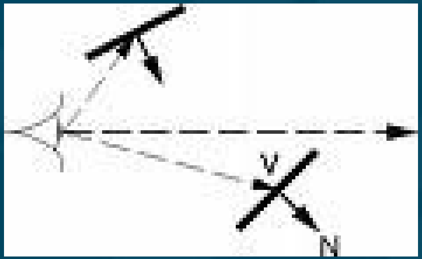
- The vertex processor is also responsible for assembling vertices into primitives:
  - Lines/curves, triangles/polygons/surfaces
- Uses the face list to index into the vertex list



Clipping and culling

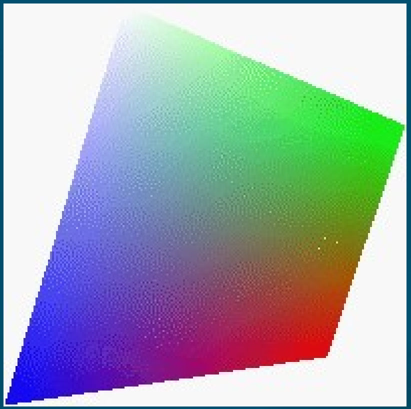


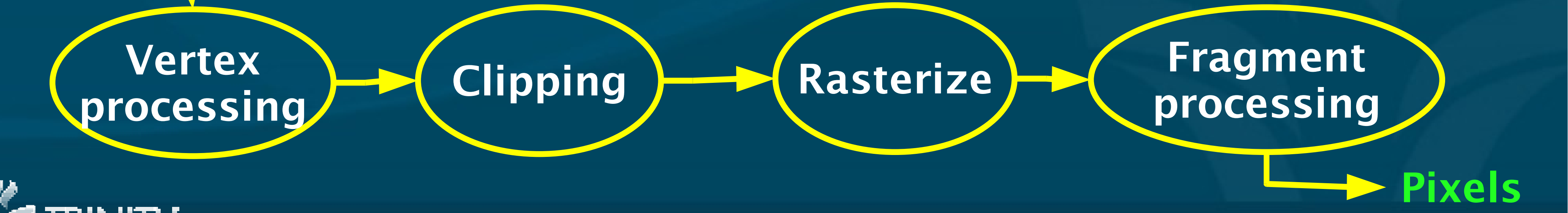
- Don't render what we can't see
- Clipping
  - Remove primitives outside of the camera's view frustrum
- Backface culling
  - Remove triangles facing away from camera
  - Usually cuts down # of triangles by about 50%!
- Other optimizations also possible



Rasterization

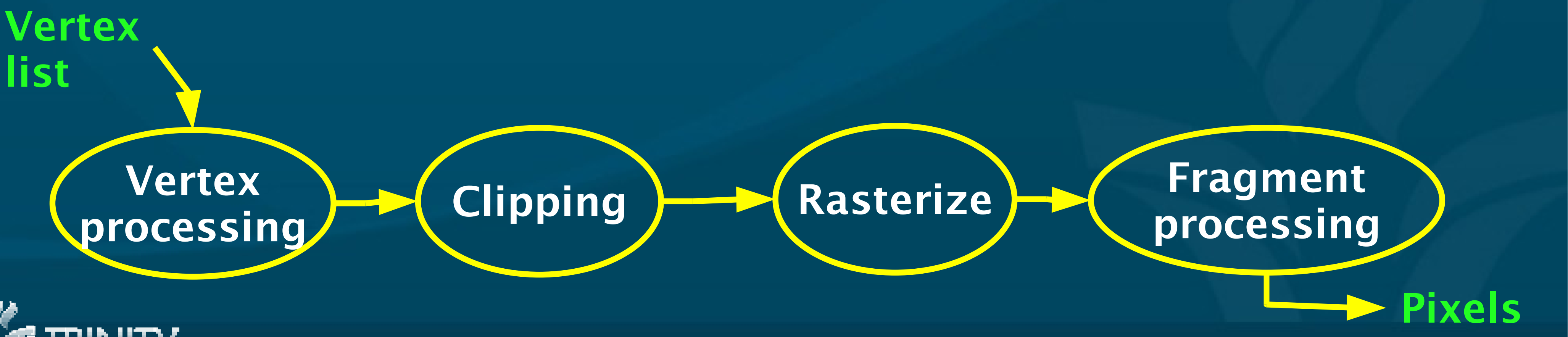
- Convert a primitive into a fragment:
  - Set of pixels just for that primitive
  - Each pixel has RGB colour and depth
  - Interpolate vertex colours over the fragment





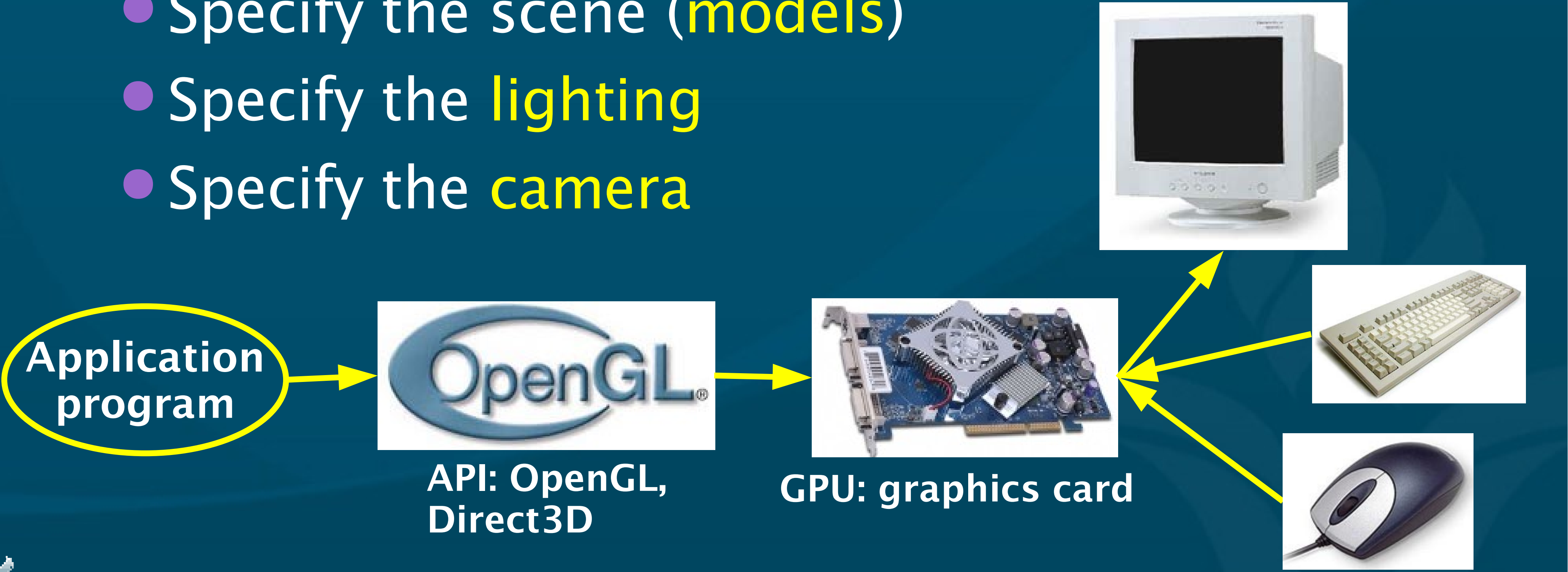
Fragment processing

- Assemble the fragments into final framebuffer
- Hidden-surface removal:
  - Some fragments may occlude parts of others
  - Handle transparency



Programmer's interface

- A graphics API allows a program to interact with the graphics pipeline
- Library subroutines (see CubeView.cxx)
  - Specify the scene (models)
  - Specify the lighting
  - Specify the camera



Application program

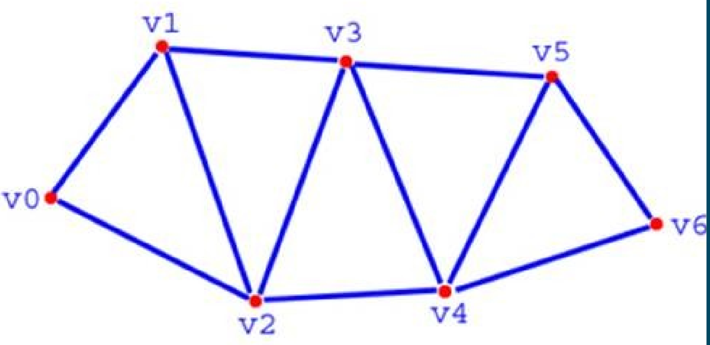
API: OpenGL, Direct3D

GPU: graphics card

Graphics API: Model

- Geometry: vertices (0D)
  - Line segments, curves (1D)
  - Polygons (2D), sometimes parametric surfaces
- Material properties: colour, specularity, etc.
- Example:

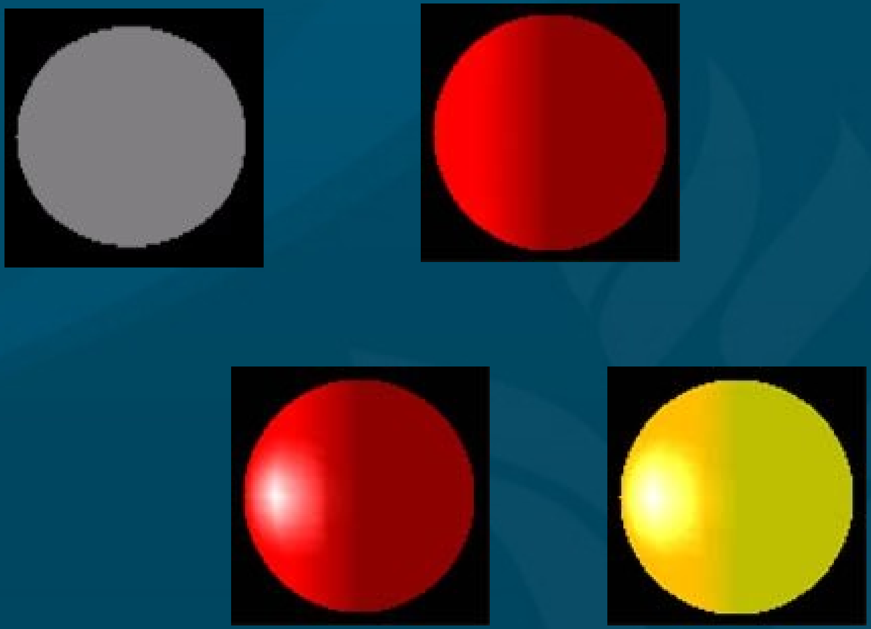
```
glBegin(GL_TRIANGLE);
    glColor3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
glEnd();
```



GL_TRIANGLE_STRIP

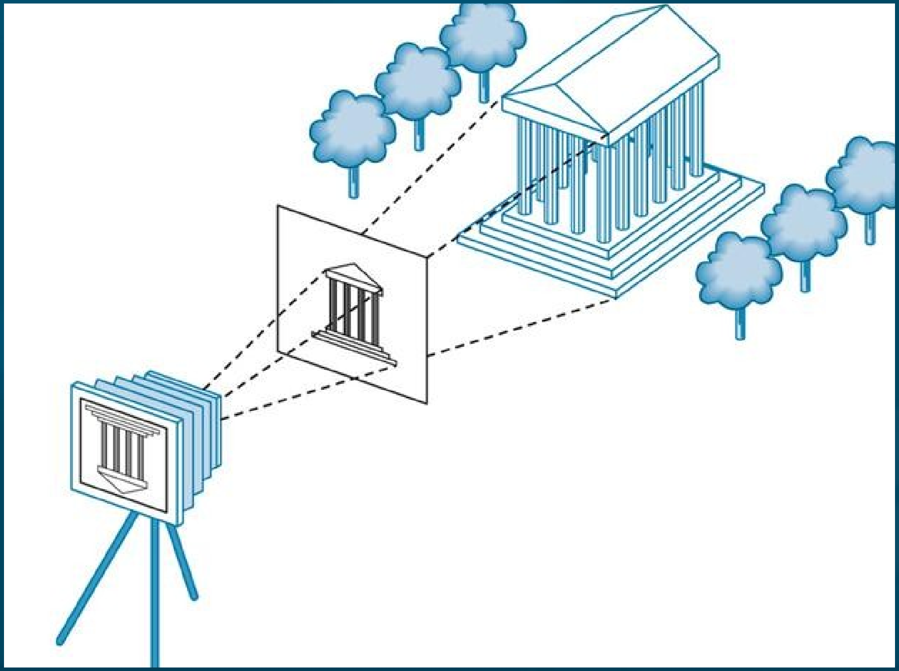
Graphics API: Lighting

- Type of light:
  - Ambient (uniform, everywhere)
  - Directional (e.g., sunlight)
  - Spotlight (cone with falloff)
  - Point vs. area light
- Material properties:
  - Ambient colour
  - Diffuse colour
  - Specular colour
  - Emissive colour



Graphics API: Camera

- 6DOF camera model:
  - Position of center of projection (3DOF)
  - Orientation (3DOF)
- Also: location and size of image plane
- Could also consider modelling lens distortion



TODO

- Lab2 due tonight
  - Design + implement your own OpenMP program
  - Lab write-up
- Midterm 1 next week Thu 15Feb
  - GUI, parallel
  - Emphasis on lecture material
  - Coding some snippets