# 4. 3D Introduction

COMP3421 Computer Graphics • KC Notes

## 4.1 Coordinates

* 3D coordinate system is **right handed coordinate system**
  + Thumb x, index y, middle z
* **Depth**: OpenGL draws objects in the order in which they are generated in code
* **Projection**: map geometry in a 3D space to a 2D image, but looks 3D to our eyes
* **Winding order**: triangles defined with counter-clockwise vertices
  + Front side has counter-clockwise vertices
* **Back face culling**: optimisation that culls non-visible triangles (back faces) before rasterization, avoids unnecessary fragments from being created
  + gl.glEnable(GL.GL\_CULL\_FACE);

## 4.2 3D transformations

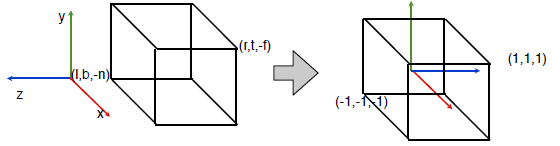
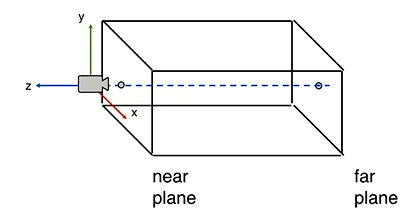
* Transformations have same structure but extra axis :
* Translation:
* Scale:
* Shear:
* Rotation: depends on axis of rotation
  + Any rotation can be decomposed into a sequence of rotations about x, y and z axes
* Right hand rule:
  + Rotation of x: y moves towards z
  + Rotation of y: z moves towards x
  + Rotation of z: x moves towards y

## 4.3 Cube

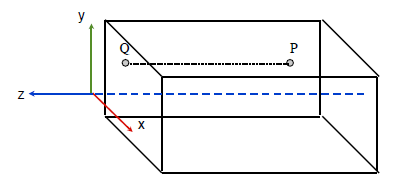
* Draw a TriangleFan square face (anticlockwise) with z = 1
* Rotate square face along Y and X axes
* Rotate coordinate frame

## 4.4 Orthographic projection

* In 2D, camera is represented by a square. In 3D, it is a **view volume**
* 3D camera can be defined as **a 3D point and orientation**, points backwards
* **View volume**: Area of space that will be displayed in the viewport
  + Objects outside are **clipped**
* Have values left, right, top, bottom, near and far
* **Project a point** from 3D view volume onto the **near plane**, and then map to the viewport
  + Projection happens **after model and view transformations**
  + Points with negative z values are in front of the camera (e.g. near = 2 puts at z = -2)
* **Depth, orthographic**: While we could map points directly to the near plane, we want to **retain depth information** – projection matrix should map z-values of visible points to be between -1 (near) and 1 (far)
  + **CVV: canonical view volume, with** clipping coordinates, scaled to the range (-1, 1) with z-axis flipped
  + Preserves parallel lines, describes shapes completely and exactly



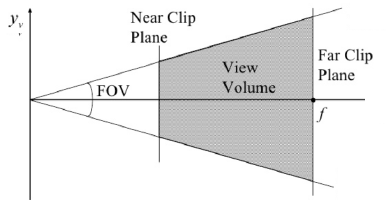
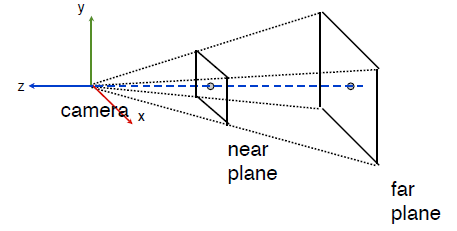
* Map x, y points using , :



* This transformation is mapped via the **orthographic transformation matrix**
  + the left and right, the top and bottom, the near and far planes
  + For Matrix4.orthographic(-1, 1, -1, 1, 1, 10); only z changes

## 4.5 Perspective projection

* **Foreshortening**: things appear smaller as they are further away
  + Biology of the eye – objects hit the pupil at different locations
* **Frustum**: view volume like a square pyramid with top cut off
  + Points get mapped to the near plane, objects far away get smaller when mapped
  + Matrix4.frustrum(-1, 1, -1, 1, 1, 10);



* Projection is easier to represent with near, far planes, and:
  + fovy, the vertical **field of view**: , usually 60
  + aspectRatio,
* Map x, y point using
* Map depth using **pseudodepth**, a non-linear value mapped between -1 and 1:
  + Objects far away do not change size as much, approximate close things better
* **Generalise homogenous coordinates**, 4th component of a point can be any value
  + Allows us to apply some non-linear operations to our point
  + Transformation is not affine (results in non-parallel lines)
* Combining perspective transformation with CVV (use right one):

## 4.6 Final Steps

* With CVV coordinates, **perspective division step** – divide by 4th component to make it 1
* **Viewport transformation**: scaled into window coordinates corresponding to pixels on screen
  + Maps pseudodepth from (-1, 1) to (0, 1)

## 4.7 Final pipeline

1. From point
2. Extend to **homogeneous coordinates**
3. Multiply by **model matrix** to get world coordinates
4. Multiply by **view matrix** to get camera coordinates
5. Multiply by **projection matrix** to get CVV coordinates
6. **Clip** to remove points outside CVV
7. **Perspective division** to eliminate 4th component
8. **Viewport transformation** to window coordinates

## 4.8 OpenGL vertex shader

|  |  |
| --- | --- |
| // Incoming vertex position  in vec3 position;  uniform mat4 model\_matrix;  uniform mat4 view\_matrix;  uniform mat4 proj\_matrix;  void main() {  // Global position in homogenous coordinates  vec4 globalPosition = model\_matrix \*   vec4(position, 1);  // Position in camera coordinates  vec4 viewPosition = view\_matrix \* globalPosition;  // Position in CVV coordinates  gl\_Position = proj\_matrix \* viewPosition;  // OpenGL will do steps 6-8  } | |
|  |  |

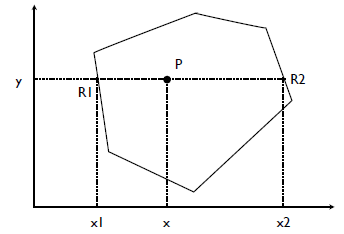
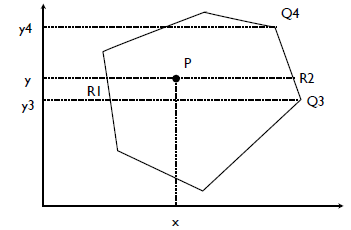
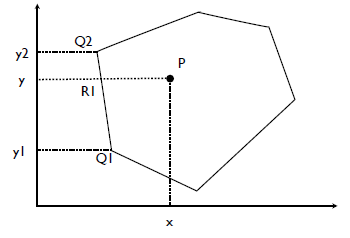
## 4.9 Fragments

* **Rasterisation**: converting triangles into collections of fragments
  + Vertex is cheap, fragment is expensive
  + Culled triangles are discarded (e.g. back-face culling)
  + Fragment corresponds to a single image pixel
  + Fragment may not make it to final image if it is discarded by depth testing
* **Hidden surface removal**: discard fragments that shouldn’t be seen
  + Make sure primitives are drawn in the right order on the **model level**, not OpenGL
  + Use **depth buffer** on the **fragment level**
* **Painter’s algorithm**: sort primitives by depth, draw from back to front, sort is expensive
  + Triangles do not have a single z value, items may overlap each other – BSP trees
    - Recursively divide world into polygons
* **Depth buffer**: keep per-pixel depth information, store in a depth buffer
  + Initially cleared to 1, polygon drawn fragment by fragment
  + If closer (depth less than 1), update pixel in colour buffer, update buffer with pseudodepth

|  |  |
| --- | --- |
| // Turn on the depth buffer  gl.glEnable(GL.GL\_DEPTH\_TEST);  // Clear the screen with the defined clear color  gl.glClear(GL.GL\_COLOR\_BUFFER\_BIT | GL.GL\_DEPTH\_BUFFER\_BIT); | |
| initialise db[x][y] = 1 for all x,y  for each triangle:  for each fragment (px,py):  d = pseudodepth of (px,py)  if (d < db[px][py]):  draw fragment  db[x][y] = d | |
|  |  |

## 4.10 Pseudodepth, bilinear interpolation

* Interpolate R1 and R2 using y axis
* Then with those two values, interpolate P using x axis



## 4.11 Problems with depth

* **Z-fighting**: depth buffer has limited precision (usually 16 bits)
  + If almost parallel, small rounding errors causes fighting
  + Using a small offset causes a gap when looked at sideways
* glPolygonOffset: draws normally but only offset depth buffer
* gl.glEnable(GL.GL\_POLYGON\_OFFSET\_FILL);
* gl.glPolygonOffset(factor, units); calculated as
  + m the depth slope of the polygon, r the smallest resolvable difference in depth
  + Usually 1, 1 for backwards, or -1, -1 for forwards