Viewing

- Now that objects are positioned in the world, we need a way to view a scene
- We break this process into two steps:
  - placing the camera/viewer
  - "taking a picture": projection

Pinhole Camera

- We start with understanding how a “real” pinhole camera works
- Light rays travel from objects through the pinhole into the back of the camera (sensors/film)
- Parameters:
  - dimensions of sensor array
  - distance between pinhole and sensors
- Image formed is upside down and backwards

Orthogonal Viewing

- Instead of a pinhole camera, each sensor in the array just looks "forward" and record what it sees.
- i.e. Light rays travel from objects to the sensor array at 90 degrees
- It is not realistic but it has many useful applications
- Distances and angles parallel to the sensor array are preserved
- Often used in technical drawings

Perspective Viewing

- For realistic viewing: objects farther away are smaller, objects closer are larger
- Use pinhole camera model
- Trick: pretend the sensor array is in front of the pinhole so the image is not upside down and backwards
- Calculations of projection make use of similar triangles
- As objects move further, perspective viewing becomes closer to orthogonal viewing

Model-View-Projection

- Place objects: done by transformation matrices (rotation, scale, translation). This is called a Model matrix.
- Position the camera: a View matrix that transforms object coordinates into camera coordinates.
- A Projection matrix is then used to transform objects from camera coordinates to clip coordinates. Only coordinates within $[-1, 1]$ are displayed.
- This is the model-view-projection approach. In practice, model and view matrices are often premultiplied to obtain a model-view matrix.

Camera Positioning

- First we define the camera coordinate system:
  - the camera is positioned at the origin $(0, 0, 0)$
  - it looks towards the negative z-axis
  - the positive y-axis is "up"
- By default, all coordinates in $[-1, 1]$ are visibile.
- It is possible to see objects "behind" the camera by default.

Camera Positioning

- If we want to put the camera in some other position and orientation, the scene has to be transformed to get them into camera coordinates.
- e.g. If we want to move the camera to position p, translate the scene by $-p$).
- So the camera is not moved, but the scene is—the view matrix is applied to vertices of objects.
- If we want to view in direction specified by a vector $\vec{v}$ (instead of $(0, 0, -1)$), a rotation is needed to rotate $\vec{v}$ to $(0, 0, -1)$.
- If we want the up direction to be $\vec{u}$, we need to rotate $\vec{u}$ to $(0, 1, 0)$.
- The view matrix would be a multiplication of the translation matrix by the required rotation matrices (on the right).
- Notice that the transformation appears backward because we move the scene, not the camera.

LookAt Transformation

- The LookAt transformation is a convenient way to specify camera position. It is defined by:
  - eye/camera position: *eye*
  - a point to look at: *at*
  - an up vector $\vec{v}_{up}$ (does not have to be parallel to viewing plane)
- The viewing direction is defined by $\vec{v}_n = eye - at$. This is normalized to $\vec{n} = \frac{\vec{v}_n}{|\vec{v}_n|}$. This is normal to the viewing plane.
- Compute

$$\vec{u} = \frac{\vec{v}_{up} \times \vec{n}}{|\vec{v}_{up} \times \vec{n}|}$$

$\vec{u}$ is orthogonal to both $\vec{n}$ and $\vec{v}_{up}$

- Compute the “up” vector orthogonal to both $\vec{n}$ and $\vec{u}$:

$$\vec{v} = \frac{\vec{n} \times \vec{u}}{|\vec{n} \times \vec{u}|}$$

- Now we have a coordinate system defined by three axes $\vec{u}$, $\vec{v}$ and $\vec{n}$.

LookAt Transformation

- Assume for now that the camera is located at the origin.
- The change of coordinate matrix from uvn to xyz is:

$$A = \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- If the camera is positioned at (x, y, z), then the transformation from camera coordinates to object coordinates is:

$$T(x, y, z)A$$

LookAt Transformation

- To convert from object coordinates to camera coordinates, the final LookAt Transformation is

$$\begin{aligned} V &= (T(x,y,z)A)^{-1} \\ &= A^{-1}T(-x,-y,-z) \\ &= A^{T}T(-x,-y,-z) \\ &= \begin{bmatrix} u_x & u_y & u_z & -xu_x - yu_y - zu_z \\ v_x & v_y & v_z & -xv_x - yv_y - zv_z \\ n_x & n_y & n_z & -xn_x - yn_y - zn_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Orthographic Projection

- When towards the negative z-axis, orthographic projection simply means removing the z coordinate:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- However, we need to also determine what can be viewed and what will be clipped out.

Orthographic Projection

- The clipping volume is a rectangular prism aligned with the axes:
  - left $\leq x \leq$ right
  - bottom $\leq y \leq$ top
  - -far $\leq z \leq$ -near (note the negative sign)
- A transformation is needed to normalize this prism into the standard viewing prism ($x, y, z \in [-1, 1]$)

Orthogonal Projection

- First, translate centre of prism to origin:
  $T = T(-(right + left)/2, -(top + bottom)/2, (far + near)/2)$.
- Then scale it to the right size:
  $S = S(2/(right - left), 2/(top - bottom), 2/(near - far))$.
- Final matrix is

$$N = ST = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{left+right}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection

- In perspective projection, we model a pinhole camera.
- The viewing plane is in front of the camera.
- Suppose the center of the camera (the pinhole) is located at the origin.
- The viewing plane is defined by $z_p = d < 0$. The distance to the viewing plane from the origin is $-d$.
- Using similar triangles, we see that a point at (x, y, z) will project to $(x_p, y_p, z_p) = \left(\frac{x}{z/d}, \frac{y}{z/d}, d\right)$.
- But this operation is nonlinear (divide by z) and cannot be represented as a matrix...
- Notice that as z increases (further away), the projected coordinates get smaller.
- Aside: what happens if a point has $z = 0$?

Perspective Projection

- To perform perspective projection as a matrix, we need to reconsider our representation of points as homogeneous coordinates.
- Instead of $w = 1$, we allow w to have any value, so the point (x, y, z) can be represented as (wx, wy, wz, w) provided $w \neq 0$.
- This can be interpreted as a line in 4-dimensional space representing each point in 3-dimensional space.
- The "true" point can be obtained by a perspective division of w.

Perspective Projection

- Back to

$$(x_p, y_p, z_p) = \left(\frac{x}{z/d}, \frac{y}{z/d}, d\right).$$

- We can write in matrix form:

$$\begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- So after perspective division by $w = \frac{z}{d}$ we obtain the desired projection.

Perspective Projection

- So the required procedure for perspective projection is:
  - multiply coordinates by

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

  - divide by the w component
- An added advantage is that the z coordinate is not lost until perspective division is done.

Perspective Projection

- So far we have not considered clipping.
- There are two ways to specify a clipping volume:
  - Frustum: left, right, top, bottom, near, far
  - Field of view: angle, aspect ratio, near, far

Perspective Projection (by Frustum)

- left, right, top, and bottom are defined in terms of near clipping plane.
- Transformation needed to transform the frustum into a cubic clipping volume $[-1, 1]$.
- Final matrix (see textbook for derivations)

$$M = \begin{bmatrix} \frac{2 \cdot near}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2 \cdot near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & \frac{-2 \cdot far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Perspective Projection (by Field of View)

- Use frustum:
  - left = −right
  - bottom = −top
  - top = near · tan(fovy)
  - right = top · aspect

Hidden Surface Removal

- One can use object-space algorithm to determine which objects are in front and ignore objects that are not visible.
- But it is easier to work in image space and simply render all objects.
- If the z coordinates are kept, they can be used to determine which objects are in front and visible.
- Some calculations may be wastsed, but much easier to implement (e.g. partially visible objects).

Depth Buffer

- Also called z-buffer.
- A framebuffer is used to store the z value of what is visible for each pixel.
- Initialize to negative "infinity" (remember z is usually negative)
- When a pixel in an object is rendered, its z coordinate is compared to the value in the z-buffer. The pixel is only rendered (and z-buffer updated) if it is closer to the viewer.
- Turn on depth buffer with `GLUT_DEPTH` in `glutInitDisplayMode`.
- Also `glEnable(GL_DEPTH_TEST)` and remember to clear. the depth buffer at each step with `glClear` and `glClearDepth`.
- You may also want to look at `glDepthFunc` if you want to change depth testing (which ones are visible).

Face Culling

- Culling means removing objects before rasterizer to save computations.
- When we define triangles, we can define the vertices in clockwise or counterclockwise order.
- By default, CCW is “front”, CW is “back”.
- Using `glEnable(GL_CULL_FACE)` and `glCullFace` allows certain orientations to be removed from the pipeline.
- Use `glFrontFace` if you want to change the default orientation for front and back.