Projection

General, transform points in $n$-space into a coordinate system of dimension less than $n$.

In computer graphics, normally transform points from 3-space into 2-space, computer screen.

"Project" rays from a center of projection through each point (vertex) in an object, compute rays' intersection with a projection plane.

If COP is finite
distance from
projection plane,
perspective projection
with perspective

Fore shortening (object appears smaller the farther it is from the viewer).

If COP at infinity,
parallel projection
(projection rays parallel along direction of projection).

Orthographic, DOP and normal to projection plane are parallel. Otherwise, oblique projection.
Normalized perspective projection

Assume projection plane normal to z-axis at \( z = 0 \),
\( \text{Cop} \in \{ z = d \} \).

By similar triangles:
\[
\frac{x_p}{d} = \frac{x}{d-z} \\
x_p = \frac{dz}{d-z} = \frac{x}{1-\frac{z}{d}}
\]

Similarly, \( \frac{y_p}{\frac{z}{d}} = \frac{y}{1-\frac{z}{d}} \)

\[\therefore \text{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{d} & 1 \end{bmatrix}\]

\[\text{P'} = \text{M} \cdot \text{P} = \text{M} \cdot \begin{bmatrix} \frac{z}{d} \\ \frac{y}{d} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ \frac{y}{1-\frac{z}{d}} \\ 1-\frac{z}{d} \end{bmatrix}\]

So in Cartesian coordinates
\[\text{P'} = \begin{bmatrix} \frac{x}{1-\frac{z}{d}} \\ \frac{y}{1-\frac{z}{d}} \\ 0 \end{bmatrix}, \text{ exactly as we want.}\]
The term \(-\frac{3}{4}\) represents perspective.

For shortening, as distance from viewer increases, \(z\) becomes more negative, so \(-\frac{3}{4}\) becomes more positive, so \(1 - \frac{3}{4} = 1 + (-\frac{3}{4})\) becomes more positive, so projection point's position on plane becomes smaller, exactly like we expect.

Applies only when COP on +z-axis and projection plane is normal to z-axis \(Ez = \pi\)

Normalized Orthographic Projection

Assume projection plane normal to z-axis \(Ez = \pi\)

Map \(P(x, y, z)\) to position \((x, y, \pi)\)

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0
\end{pmatrix}
\]

\[P' = M \cdot P = M \cdot \begin{bmatrix}
x \\
y \\
\frac{z}{2} \\
1
\end{bmatrix} = \begin{bmatrix}
x \\
y \\
\pi \\
1
\end{bmatrix}\]

Again, only for projection plane \(Ez = \pi\), normal to +z-axis.
Arbitrary Location/Orientation Projections

In order to perform projection from an arbitrary view location and direction, we must first align the View Reference Coordinate System (VRC) with the WC System.

Define:

View plane defined by a view reference point (VRP) and a view plane normal (VPN)

\[ \text{VRP} \]
\[ \text{VPN} \]
\[ \text{N-axis} \]

\[ \text{Origin at VRP: one axis is VRN, called u-axis.} \]
\[ \text{User-defined up-vector (VUP) projected onto view plane to form v-axis.} \]
\[ \text{Finally, u-axis orthogonal to u, v-axes.} \]

So VRP, VPN, VUP sufficient to define VRC system and associated view plane.

Next, define Viewport window as \((u_{min}, v_{min})\) to \((u_{max}, v_{max})\), also defines center of window CW.
Finally, define a center of projection PRP and projection type (perspective or parallel).

PRP specified in VRP (u,v,w)
So PRP's position relative to VRP maintained if VRP moved.

Can specify front, back (F,B) clip planes to limit view volume.

Dop specified relative to VRP, normal to VRU.

OpenGL uses three separate commands to allow you to define your projection properties, and one command to specify the virtual camera properties.

- `gluPerspective (fov, aspect, front, back)`

Defines a perspective projection symmetric in the x and y directions. `fov` defines the field-of-view angle, in degrees, in the yz-plane (i.e., the vertical field-of-view angle): $0 < \text{fov} < 180\degree$. 

aspect is the width:height aspect ratio of the viewing frustum. East is the distance from the viewpoint to the front clip plane, along the +z-axis. West is the distance to the back clip plane, along the -z-axis.

\[ \text{viewpoint} \]

\[-\text{Frustum (left, right, bottom, top, front, back)} \]

Defines a perspective projection whose boundaries pass through (left, bottom, -front) and (right, top, -front), assuming viewpoint is at default location at origin.

\[ \text{viewpoint} \]
- glortho (left, right, bottom, top, front, back)

Defines a parallel projection with (left, bottom, -front) and (right, top, -front) mapped to the lower-left and upper-right corners of the viewport window. The direction of projection is parallel to the z-axis, in the +z-direction.

By default, the camera is positioned at the origin, looking down the -z-axis, with up aligned along the ty-axis. This can be changed using:

- gluLookAt(eye_x, eye_y, eye_z, center_x, center_y, center_z, up_x, up_y, up_z)

with the viewpoint at (eye_x, eye_y, eye_z), looking at (center_x, center_y, center_z), with the up-direction pointing in (up_x, up_y, up_z).
Some notes:

Projections are encoded as a 4x4 matrix on the GL_PROJECTION matrix stack. The stack must normally be prepped with an initial identity matrix:

```c
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluPerspective (120, 1, 1, 100);
```

The camera's position is actually encoded as a set of translations and rotations on the objects in the scene, so it belongs on the GL_MODELVIEW matrix stack:

```c
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
gluLookAt (0, 0, 5, 0, 0, 0, 0, 1, 0);
```

Note also that it is possible to define a perspective projection with glFrustum that is not symmetric in x (or y), but it may be difficult to anticipate or control the resulting images.
gluLookAt() defines the VRC system's values.

Since given

\[ \text{gluLookAt}(ex, ey, ez, Cx, Cy, Cz, uX, uY, uZ) \]

we normally assume

\[ \text{VP} = (C_x, C_y, C_z) \]
\[ \text{VPN} = (ex-C_x, ey-C_y, ez-C_z) \]
\[ \text{VUP} = (u_x, u_y, u_z) \]

The projection command fills in the
viewport location and near and far clip
plane locations:

\[ \text{glFrustum}(l, r, b, t, F, B) \]

Here:

\[ (\text{Vmin, Vmax}) = (l, b) \]
\[ (\text{Vmin, Vmax}) = (r, t) \]
\[ \text{near} = F, \text{far} = B \]

If instead:

\[ \text{gluPerspective}(\theta, \text{aspect}, F, B) \]
Then we use the following diagram to compute the corners of the viewpoint:

\[
\begin{pmatrix}
\text{Vmax} \\
\text{Vmin} \\
\text{F} \\
\text{E} \\
\end{pmatrix}
\]

And similarly for \( \text{Vmin} \) and \( \text{Vmax} \), remembering that aspect = \( \frac{W}{H} \), the ratio of the viewport's width to its height.

Finally, for an orthographic projection:

\[\text{glOrtho}(l, r, b, t, F, B)\]

Here:

\[
\begin{align*}
(\text{Vmin}, \text{Vmin}) &= (l, b) \\
(\text{Vmax}, \text{Vmax}) &= (r, t) \\
near &= F, \quad far = B
\end{align*}
\]

As with \text{glFrustum}
How does OpenGL manage arbitrary view positions and directions?

Goal: Go from general VRC system to constraints we assumed when we built projection matrix (e.g., view plane @ z = 0 normal to z-axis, center of projection on +z-axis).

1. Translate VRP to origin
2. Rotate u,v,w axis to align with WC x,y,z axes

Can be performed with one translation T and one orthonormal rotation R

So at this point M = R,T

3. Skew to correct for an oblique direction of projection in parallel case, or for a CW-PRP direction that does not lie on +z-axis for perspective projection

\[ \begin{align*}
\text{e.g.} & \quad \begin{pmatrix}
\cos\alpha \\
\sin\alpha
\end{pmatrix} \\
& \begin{pmatrix}
x(1-y) \\
0
\end{pmatrix} = 0 \\
& \begin{pmatrix}
\text{u} \text{or minimum} / \text{u} \\
\text{v} + (\text{v} \times \text{u} / \text{w}, \text{w})
\end{pmatrix}
\end{align*} \]

PRP (PRP, PRP, PRP, PRP, PRP)
Can be accomplished with a shear matrix

So now $M = SH \cdot R \cdot T$

(a) For parallel projection, translate
   $C$ assuming to be on front clip plane
   to origin, scale view volume to size
   $2 \times 2 \times 1 \ (-1 \leq x, y \leq 1, -1 \leq z \leq 1)$

So parallel projection has $M = S \cdot T \cdot SH \cdot R \cdot T$

(b) For perspective projection, perform a
    non-uniform scale to give $x, y$ bounding
    planes unit slope, scale uniformly in all 3
    axes to move back plane to $z = 1$, and
    finally translate front clip plane to origin
    and re-scale view volume to $2 \times 2 \times 1$ cube

So perspective projection has $M = S \cdot T \cdot S \cdot S \cdot SH \cdot R \cdot T$

Point is, in both cases, a single $4 \times 4$
projection matrix made up of a combination
of base transform matrices can be constructed
to map from generic view position to
constrained view position
At this point we can clip all objects to the canonical 2x2x1 view volume, then apply the simplified projection matrices $M_p$ or $M_p$ to project the visible vertices to the projection plane.