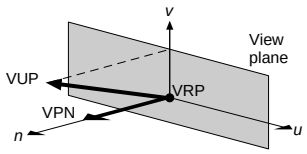
Outline

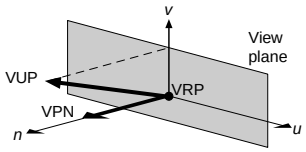
- Specification of a 3D view.
- Implementing parallel projections.
- Implementing perspective projections.
- Mapping to the view port.

Specification of 3D views



- The projection plane = view plane.
- Defined using:
  - View Reference Point (VRP), View Plane Normal (VPN)
  - and View Up Vector (VUP)

Specification of 3D views

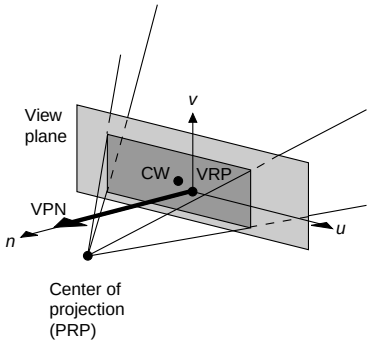


- Together define the Viewing Reference Coordinate (VRC) system.
- The VRP (point) and VPN and VUP (directions) are specified in a right handed world (application model) coordinate system.

Specification of 3D views

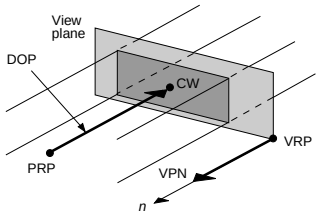
- The view window is then defined by $(u_{\min}, v_{\min})$, $(u_{\max}, v_{\max})$.
- The centre of projection and Direction Of Projection (DOP) are defined by a Projection Reference Point (PRP) and an indicator of the projection type:
- The coordinates of the PRP are defined in the VRC.

Perspective projection



Specification of 3D views

- Parallel projection



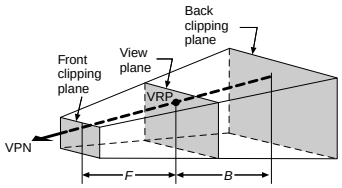
- If the projection is parallel, the the DOP is defined by the direction between the PRP and the centre of the projection window.

Specification of 3D views

- It is simpler for the programmer to change the direction of projection required (rotate / move the camera), at the expense of extra complexity if the PRP is moved (i.e. to get different views of the object – change the lens).
- OpenGL uses a very similar method to specify the view volumes – we will come back to this.
- The camera analogy is very powerful and general.

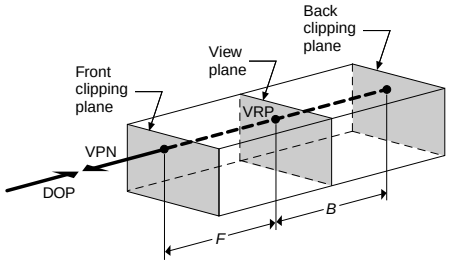
Specification of 3D views

- Finite view volumes are defined by selecting the signed quantities F and B which define the locations of the front and back clipping planes.
- Both these planes are parallel to the view plane, thus F and B are defined along the VPN.



Specification of 3D views

- We want a finite view volume because we do not want to draw objects close (or behind us) and objects which we cannot see.



Specification of 3D views

- To display the contents of the view volume the objects are mapped into a unit cube whose axes are aligned with the VRC system (called the Normalised Projection Coordinates (NPC)).
- Objects which are far away are squashed more (small)!
- To create a wire-frame display of the contents of the 3D viewport we can simply drop the z coordinate.

Implementing 3D→2D projections

- Define normalising transformations N_{par} (parallel) and N_{per} (perspective) that transform the points in world coordinates within the view volume to points in the normalised projection coordinates.
- Can apply clipping and hidden line removal to these canonical view volumes.
- Next section gives the details necessary to implement (and understand?) planar geometric projections.

Parallel projections

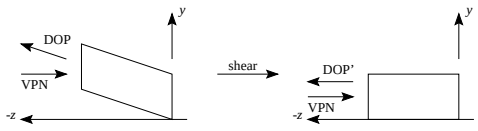
- Canonical view volume is the unit parallelepiped defined by the planes, $x = -1$, $x = 1$, $y = -1$, $y = 1$, $z = 0$ and $z = -1$. The steps necessary to achieve a canonical view volume are
  1. Translate the VRP to the origin.
  2. Rotate the VRC such that it is aligned with the world coordinate system.
  3. Shear (in x and y) so that the direction of projection is aligned with the z axis.
  4. Translate and scale into the canonical view volume (parallelepiped).

Parallel projections

- Steps 1 and 2 combined produce the view orientation matrix (`GL_MODELVIEW`), while steps 3 and 4 give the view mapping matrix (`GL_PROJECTION`).
- Steps 1 and 2 are logical.
- BUT ....

Parallel projections

- Step 3, shearing (so that z value represents distance from the viewer:



- Step 4 is not necessary for display, however most clipping algorithms require a canonical volume element.
- $N_{par} = S_{par}T_{par}H_{par}RT(-VRP)$

Perspective projections

- Steps 1 and 2 are the same as those in the parallel case but an additional step brings the PRP to origin.
  1. Translate the VRP to the origin.
  2. Rotate the VRC such that it is aligned with the world coordinate system.
  3. Translate so that the centre of projection (i.e. the PRP) is at the origin.
  4. Shear (in x and y) so that the direction of projection is aligned with the z axis.
  5. Scale into the canonical view volume (truncated pyramid).

Perspective projections

- The final composite transformation matrix is given by $N^*_{per} = S_{per}H_{par}T(-PRP)RT(-VRP)$.
- N_{par} and N^*_{per} will not affect the homogeneous coordinate w. However the additional step to the computation of N_{per} transforming the truncated pyramid to a parallelepiped will:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{min}} & \frac{-z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

- $N_{per} = MN^*_{per}$.

Clipping and Projection to 2D

- Clipping is usually carried out in the canonical view volume since the algorithm will be independent of the projection type.
- Projecting the 3D canonical volumes to 2D is very simple, we just retain the x and y coordinates.
- The matrix to do this is just:

$$M_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ or } M_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}.$$

3D Viewport transformation

- This is similar to the 2D case.
- The objects transformed by the projection will then be transformed into the viewport coordinates using the following matrices:

$$\begin{bmatrix} 1 & 0 & 0 & x_{vmin} \\ 0 & 1 & 0 & y_{vmin} \\ 0 & 0 & 0 & z_{vmin} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x_{max}-x_{vmin}}{2} & 0 & 0 & 0 \\ 0 & \frac{y_{max}-y_{vmin}}{2} & 0 & 0 \\ 0 & 0 & \frac{z_{max}-z_{vmin}}{1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- To plot the resulting 2D object we divide by w and then simply ignore the z coordinate and plot the x and y coordinates.

Summary

- To summarise the process of 2D viewing of 3D objects is:
  - 3D → homogeneous,
  - apply N_{par} or N_{per},
  - homogeneous → 3D,
  - clip,
  - 3D → homogeneous,
  - project using M_{ort} or M_{per},
  - transform into device coordinates (window to viewport): homogeneous → 2D.

Viewing in OpenGL

- OpenGL viewing definition uses the camera analogy.
- Two matrices define the total projection:
- `GL_PROJECTION` defines the projection using the matrices: $MS_{per}H_{par}$ as given in the notes.
- Use the command `glFrustum` to set the viewing parameters.
- Define the viewing window (x_{min}, y_{min} and x_{max}, y_{max}), and the near and far clipping planes.
- Think of this like the lens of a camera.

Viewing in OpenGL

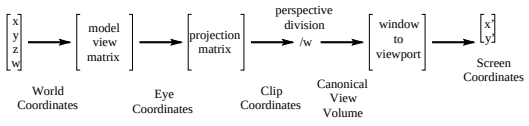
- The other matrix to set is the `GL_MODELVIEW` matrix which controls both the objects and the view.
- Use `gluLookAt` with the eye, location to look at and up vector to define the view of the object.
- This is like the matrices $S_{par}T_{par}$ as given in the notes.
- Think of this like aiming the camera.

Practical viewing in OpenGL

- Easy to get lost in space: keep the far clipping plane to a large value – changing the near clipping plane changes the degree of perspective distortion.
- Use asymmetric (x_{min}, y_{min} and x_{max}, y_{max}) to achieve false perspective – make VPN non-parallel to DOP.
- Start with a large front clipping plane: (x_{min}, y_{min} and x_{max}, y_{max}) – then focus in on the object.
- Set the viewport using `glViewPort` and give the origin and width / height (keep the same as the aspect ratio of the front clipping plane.
- If it ain't broke don't fix it!

Viewing in OpenGL

- The OpenGL viewing pipeline looks like:



- For both projection and model view matrices use `glMatrixMode` to define which to use and then don't forget to initialise them using `glLoadIdentity`.

Viewing in OpenGL

- Can also use parallel projections: use `glOrtho` to set the projection matrix.
- The model view matrix is set in the same way as before.
- Use `glLoadMatrix` to define our own projection matrices (masochists only).
- Beware the difference between modelling and viewing transformations.
- Viewing transformations are always set first in the code, since they are applied last to the animated models.

Lighting OpenGL

- OpenGL uses simplified models to compute lighting. This is a complex issue – it is covered in the lectures, where I will discuss its use.
- Main thing is we need to set the material properties (with respect to the different lighting types: ambient, diffuse and specular). We then need to define the lights – position and colour.
- To compute lighting we must know the surface normals, as well as vertex location and material properties which can be different on different faces (front and back).
- Need to initialise and use the depth buffer to get proper 3D effect.

Animation with OpenGL

- For this we are constrained by GLUT– it is quite particular about animation.
- Main tool to use is to set the `glutIdleFunction` – whenever GLUT has processor time it runs the specified function.
- Can pass in the `NULL` function (no animation).
- There can only ever be one `glutIdleFunction` so this must process all the animation instructions and then call `glutPostRedisplay` which sets a redraw flag.
- There are several methods we can use for animation: remember to use `GLUT_DOUBLE` mode, and `glutSwapBuffers`.

Methods of animation in OpenGL

- Animation is about change over time.
- Most natural method is to employ the `glutIdleFunction` to increment time and make what is drawn depend on time – solar system example.
- But we could directly change the transformation matrices, without worrying about their dependence on time – the house spinning example.
- Or we could use procedural animation and call functions to do higher level things, like open or close jaws – the robot arm example (but note GLUT is not designed to work this way).

Summary

- Having finished this lecture you should:
  - understand how to implement projection in computer graphics;
  - contrast perspective and parallel projections;
  - be able to set the projections in OpenGL;
  - understand how OpenGL can be used for animation.
- The course gets no harder than this, I think!