Viewing transformations

October 14-21
OpenGL transformation pipeline

Four main stages:

- **Modelview**: object coords to eye coords
  \[ p_{\text{eye}} = M p_{\text{obj}} \]
  \((x_{\text{obj}}, y_{\text{obj}}, z_{\text{obj}}, w_{\text{obj}}) \rightarrow (x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}}, w_{\text{eye}})\)
  in eye coordinates, the view direction is along negative z,
  the camera is at the origin

- **Projection**: eye coords to clipping coords
  \[ p_{\text{clip}} = P p_{\text{eye}} = P M p_{\text{obj}} \]
  \((x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}}, w_{\text{eye}}) \rightarrow (x_{\text{clip}}, y_{\text{clip}}, z_{\text{clip}}, w_{\text{clip}})\)
  in clip coordinates, the viewing volume is a box with x,y,z dimensions [-1,1]
  Perspective division: homogeneous (4d) clipping coords coords to 3d
  normalized device coords
  \((x_{\text{clip}}, y_{\text{clip}}, z_{\text{clip}}, w_{\text{clip}}) \rightarrow (x_{\text{ndc}}, y_{\text{ndc}}, z_{\text{ndc}}) = (x_{\text{clip}}/w_{\text{clip}}, y_{\text{clip}}/w_{\text{clip}}, z_{\text{clip}}/w_{\text{clip}})\)

- **Viewport**: normalized device coords to pixel coordinates + depth
  simple rescale and shift of x and y, to take them into the [0,pixel_width] and
  [0,pixel_height] ranges respectively; additionally, z is rescaled to [0..1]
  (default) or a different range set by glDepthRange

©2004, Denis Zorin
OpenGL transformation pipeline

Modelview matrix

- combines object positioning in the world coords and camera positioning
- camera positioning can be regarded as moving the world with respect to the camera

©2004, Denis Zorin world/obj. coords  eye coords
OpenGL transformation pipeline

Projection matrix

- rescales the viewing volume (frustum) to $[-1,1]$ in $x,y,z$
- projection to the image plane becomes simple (discard $z$)
- $z$ is retained for $z$-buffering
- NDC are almost the same as clip: $(x/w,y/w,z/w)$ instead of $(x,y,z,w)$
OpenGL transformation pipeline

Viewport

- convert x and y from normalized to pixel coords, rescale z
- \( x_{\text{pixel}} = 0.5*(x_{\text{clip}}+1) \times \text{viewport_width} + \text{viewport_xoffset} \)
- \( y_{\text{pixel}} = 0.5*(y_{\text{clip}}+1) \times \text{viewport_height} + \text{viewport_yoffset} \)
- \( \text{depth} = 0.5*(z_{\text{clip}}+1) \)
  
  the formula for depth can be changed using \text{glDepthRange(min_depth,max_depth)} to be
  
  \( 0.5*(z_{\text{clip}}+1) \times (\text{max_depth}-\text{min_depth}) + \text{min_depth} \)

min_depth and max_depth should be in [0..1]

©2004, Denis Zorin
Camera specification

Define the dimensions of the viewing volume (frustum)

- most general `glFrustum(left, right, bottom, top, near, far)`

In the picture:
- l = left
- r = right
- b = bottom
- t = top
- n = near
- f = far
- s = far/near

©2004, Denis Zorin
Camera specification

Less general but more convenient:

```
    gluPerspective(field_of_view, aspect_ratio, near, far)
```

In the picture:

- `fov` = field of view,
- `h/w` = `a` = aspect ratio

Relationship to frustum:

- `left` = `-a * near * tan(fov/2)`
- `right` = `a * near * tan(fov/2)`
- `bottom` = `-a * near * tan(fov/2)`
- `top` = `a * near * tan(fov/2)`

`gluPerspective` requires `fov` in degrees, not radians!
Camera positioning

- If it is preferable to regard the camera as movable, positioning is achieved by computing a transformation $M$ that moves the standard camera (at zero, looking in negative $z$ axis, with up direction along $y$ axis) to the desired position and orientation and then applying $M^{-1}$ to the world.
- This is done by `gluLookAt`.
Camera positioning

- `gluLookAt(eye_x, eye_y, eye_z, center_x, center_y, center_z, up_x, up_y, up_z)`
- up vector should not be parallel to eye-to-center vector
- up vector need not be perpendicular to eye-to-center, but `gluLookAt` will force it to.
Camera positioning

- Let $e$ be the eye (camera) position, $c$ the “lookat” point (center), $v = (c - e)/|c - e|$;
- make a “left” vector perp. to $u$ and $v$: $L = v \times u/|v \times u|$;
- new up vector, perp. to $v$: $u’ = L \times v$;
- rotation taking unit axis vectors to $L, u’,-v$, establishing correct orientation:

$$
M = \begin{pmatrix}
L_x & u'_x & -v_x & 0 \\
L_y & u'_y & -v_y & 0 \\
L_z & u'_z & -v_z & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

- this needs to be followed by translation to $e$, to get correct position
Camera positioning

- Positioning the camera as an object means calling `glTranslatef(ex, ey, ez)`, followed by `glMultMatrix(M)`; equivalent to setting modelview to $TM$

- Positioning *the world* w.r.t. to the camera (and this is what the modelview matrix does), means using the inverse matrix $(TM)^{-1}$, equivalent to $M^T (-T)$, because inverse of rotation is transpose of that rotation and inverse of translation $T$ is $-T$

- Finally we get
  
  `glMultMatrix(M); glTranslatef(-ex, -ey, -ez)`