Quiz 1

Victoria Deng

August 2020

Lecture 2 - Perspective and Polygonal Geometry

Pinhole Camera Model: A simple setup where there is a hole on one side that projects an image. In reality would be reflected upside down, however for the purposes of computer graphics, we project the image in front of the object.

Modelling a Virtual Camera:

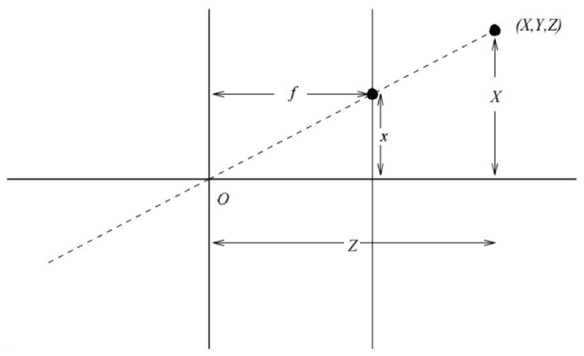

Figure 1: Perspective diagram of virtual camera geography, X is the object being projected

Perspective Formulas: Applicable only for camera-centred coordinates.

$$\frac{x}{f} = \frac{X}{Z}$$
$$\frac{y}{f} = \frac{Y}{Z}$$

Virtual Camera Geometry: The image project surface is imagined to be in front of a projection centre, the projected image would be geometrically equivalent to the object's dimensions.

Perspective Projection: Requires width, height and length to draw a 3D image. Depth is also conveyed depending on the number of point-perspectives required.

Horizon line: Key component of perspective modelling, the horizon line is considered eye level.

Vanishing Points: Point in linear perspective, the point where all vanishing lines should converge. Generally placed on the horizon line for 1 and 2 point perspectives.

Vanishing Lines: Lines that meet at the vanishing point on the horizon line.

X Point Perspective: The number of vanishing points in a model. Three point perspective has the vanishing point below the horizon line

Polygon: Any plane figure bounded by straight line segements. Can be in the form of a polygonal arc (polylines or an open polygon), polygonal boundaries (a closed polygon) or filled polygons.

Representation of Polygons: Can be represented as a set of line segments (connected but unordered), or an ordered sequence of vertices using absolute or relative coordinates. The ordered sequence can allow for 'lazy rendering', as if part of a polygon is ont shown, then it doesn't need to be rendered. Another convention is walking order, where there is a sequence of vertices on the outside of a polygon and the inside of one. Generally the sequences are opposites, eg anti-clockwise for outside and clockwise for inner.

Polygon Types: Convex (All edges under 180 degrees), Concave (One edge above 180 degrees), Non-Simple (Two or more connected polygons), Star, Multiple Boundary (Hollow polygons inside a polygon).

Level of Detail: Affected by the amount of polygons required to render an object. More polygons equals more details, however when using a model we need to balance the detail and the expense of rendering it. For instance, if we want to model a rabbit 100m away, is it necessary to use the most detailed model?

Summary: Polygons are used a lot for rendering, where the most common polygon used is a triangle as every other polygon is basically a bunch of triangles.

Lecture 3 - The Rendering Pipeline (Direct3D)

Pipeline Stages: Represented in the order that they are executed in the Direct3D pipeline

Input Assembler: Building block stage, main purpose is reading in the data from our buffers then transforming it into a primitive format to be used by the rest of the pipeline. Generally use triangle lists for defining triangles rather than individually defining vertices.

Vertex Shader: Performs operations on indivual vertices from the Input Assember, typically involves transforming them or bringing them into the world/scene.

Tesselation Stages: Optional, but used to add details and smooth out triangles. Can take models with lower details and render in higher details, or generate additional vertices.

Stages: Contains 3 stages
Hull Shader Stage
Tessellator Stage
Domain Shader Stage

Geometry Shader: Optional, unlike the vertex shader it will act on an entire object rather than one vertex. Can use algorithms to alter the geometry, for example the details of shadows in a game.

Stream Output Stage: Allows us to recieve data from the geometry shader and pass it back into the pipeline for additional processing in other shaders. Good for dynamic objects like water and fire, or objects that need more detail and movement that one pass through the pipeline. We can decide/program how many passes are required. Also done on the GPU rather than CPU.

Rasterizer Stage: Converts vector information (shapes or primitives) into a raster image (pixel) for the purpose of displaying 3D graphics. Basically adds pixels onto an object. This stage also performs Culling.

Culling: Avoids rendering vertices that are not shown or displated. If a triangle cannot be seen by a camera, it will not be rendered. DirectX operates by 'Counter Clockwise Culling', meaning that triangles with vertices in a counterclockwise order will not be rendered.

Pixel Shader: Aesthetic part of the pipelines. We can colour pixels individually or use algorithms to provide light and colour to pixels. Can also render colours in a specific order if necessary.

Output Merger Stage: Combines pixel shader output values to produce a final image. Some issues that may arise is bad fps, long processing times or the screen updating before the next frame is drawn.

Double Buffering: An alternative, as we do not want to draw object directly on the screen. We can instead draw the image into a buffer, which so it can swap images using said buffer when the image is complete.

Framerate: How often the data is sent through the pipeline. 30fps is the image being sent 30 times around the pipeline per second. Generally want stable framerate over higher fps.

Lecture 4 - Camera Control

Camera Parameters:

Roll: Rotation around the front to back axis (rolling over, circle around axis pointing in/out)
Pitch: Rotation around the side to side axis (circle around the left/right axis)
Yaw: Rotation around the vertical axis (circle around the up/down axis)

Focal Length: Approximates the behaviour of real camera lens. Objects that the distance of the focal length are in focuse, while those too close or far will be blurry.

Zoom: Focusing on a section of an image and increasing its overall size to manipulate or view in greater detail. Note that if an object is rendered with very little details, zooming in will not make it clearer.

Camera Controls:

User Camera Control: Interactive approach, proposs a set of mapping between the dimensions of the user input device (eg. mouse or keyboard) and the camera parameters. Should note that while the user can control the camera, there should be some limits on world edges and terrain.
Automatic Camera Control: Application takes care of camera controls, either based on user preferences (like first or third person) or predfined heurestics that are set during implementation.
Mixture: A lot of games use a mixture of these controls.
Visualisation: Requires interactive control to explore and pursue hypotheses concerning data. Restricted to some degree depending on the points that are able to explore, but has a lot of freedom in navigating around a space (similar to exploring Space).

Viewpoints: Mainly applicable for games

First Person: Has the camera move with the mouse/user input. The most common game type for this would be a first person shooter. An issue that may arise is having more restrictions on the field of view as opposed to a third person persepective.
Third Person: Has the camera track characters from a distance, generally slightly above and behind the character. Issues that may arise is occlusion, which can be solved by not rendering the object or making it transparent. Another issue is automatically shifting if the angle is unable to view the character (behind a wall etc.)
Action Replay: Automatically changing perspective (like a cutscene or replay of events), should always have the event in view but can shift from first to third person view.

Navigation: How a user can move the camera, can be combined

Eyeball in Hand: User can manipulate the viewpoint as if it was held in their hand.
User imagines themselves moving around an environment. Can also move up and down, but like a hand has some limitations.
World in Hand: User's navigation is directly connected to the object or environment. Moves the object or world around in their hand, so has a set viewpoint (fixed camera) but interacts by rotating the object itself. Eg. Looking at a rubix cube in someone's hand, we would be moving the cube itself rather than the person.
Flying vehicle: Camera is treated as a control stick for an airplane. Relatively unconrstained and allows a user to explore a 3D space without too many limitations.
Walking: Navigates a virtual environment by allowing the user to walk. The camera would be a set height, but can rotate, height may be able to change relative to the environment
(like walking up a hill) but ultimately the height in relation to a person would not change.
(i.e your eyeballs can't change height).