2.6 Through the Pipeline

Points, lines, and triangles are the rendering primitives from which a model or an object is built. Imagine that the application is an interactive *computer aided design* (CAD) application, and that the user is examining a design for a waffle maker. Here we will follow this model through the entire graphics rendering pipeline, consisting of the four major stages: application, geometry, rasterization, and pixel processing. The scene is rendered with perspective into a window on the screen. In this simple example, the waffle maker model includes both lines (to show the edges of parts) and triangles (to show the surfaces). The waffle maker has a lid that can be opened. Some of the triangles are textured by a two-dimensional image with the manufacturer's logo. For this example, surface shading is computed completely in the geometry stage, except for application of the texture, which occurs in the rasterization stage.

Application

CAD applications allow the user to select and move parts of the model. For example, the user might select the lid and then move the mouse to open it. The application stage must translate the mouse move to a corresponding rotation matrix, then see to it that this matrix is properly applied to the lid when it is rendered. Another example: An animation is played that moves the camera along a predefined path to show the waffle maker from different views. The camera parameters, such as position and view direction, must then be updated by the application, dependent upon time. For each frame to be rendered, the application stage feeds the camera position, lighting, and primitives of the model to the next major stage in the pipeline—the geometry stage.

Geometry Processing

For perspective viewing, we assume here that the application has supplied a projection matrix. Also, for each object, the application has computed a matrix that describes both the view transform and the location and orientation of the object in itself. In our example, the waffle maker's base would have one matrix, the lid another. In the geometry stage the vertices and normals of the object are transformed with this matrix, putting the object into view space. Then shading or other calculations at the vertices may be computed, using material and light source properties. Projection is then performed using a separate user-supplied projection matrix, transforming the object into a unit cube's space that represents what the eye sees. All primitives outside the cube are discarded. All primitives intersecting this unit cube are clipped against the cube in order to obtain a set of primitives that lies entirely inside the unit cube. The vertices then are mapped into the window on the screen. After all these pertriangle and per-vertex operations have been performed, the resulting data are passed on to the rasterization stage.

Rasterization

All the primitives that survive clipping in the previous stage are then rasterized, which means that all pixels that are inside a primitive are found and sent further down the pipeline to pixel processing.

Pixel Processing

The goal here is to compute the color of each pixel of each visible primitive. Those triangles that have been associated with any textures (images) are rendered with these images applied to them as desired. Visibility is resolved via the z-buffer algorithm, along with optional discard and stencil tests. Each object is processed in turn, and the final image is then displayed on the screen.

Conclusion

This pipeline resulted from decades of API and graphics hardware evolution targeted to real-time rendering applications. It is important to note that this is not the only possible rendering pipeline; offline rendering pipelines have undergone different evolutionary paths. Rendering for film production was often done with *micropolygon* pipelines [289, 1734], but ray tracing and path tracing have taken over lately. These techniques, covered in Section 11.2.2, may also be used in architectural and design previsualization.

For many years, the only way for application developers to use the process described here was through a *fixed-function pipeline* defined by the graphics API in use. The fixed-function pipeline is so named because the graphics hardware that implements it consists of elements that cannot be programmed in a flexible way. The last example of a major fixed-function machine is Nintendo's Wii, introduced in 2006. Programmable GPUs, on the other hand, make it possible to determine exactly what operations are applied in various sub-stages throughout the pipeline. For the fourth edition of the book, we assume that all development is done using programmable GPUs.

Further Reading and Resources

Blinn's book *A Trip Down the Graphics Pipeline* [165] is an older book about writing a software renderer from scratch. It is a good resource for learning about some of the subtleties of implementing a rendering pipeline, explaining key algorithms such as clipping and perspective interpolation. The venerable (yet frequently updated) *OpenGL Programming Guide* (a.k.a. the "Red Book") [885] provides a thorough description of the graphics pipeline and algorithms related to its use. Our book's website, realtimerendering.com, gives links to a variety of pipeline diagrams, rendering engine implementations, and more.

---

1 Since a CPU itself is pipelined on a much smaller scale, you could say that the application stage is further subdivided into several pipeline stages, but this is not relevant here.

2 "Direct3D" is the three-dimensional graphics API component of DirectX. DirectX includes other API elements, such an input and audio control. Rather than differentiate between writing "DirectX" when specifying a particular release and "Direct3D" when discussing this particular API, we follow common usage by writing "DirectX" throughout.