Rendering

Rendering Pipeline

CS 415: Game Development

Professor Eric Shaffer

ILLINOIS

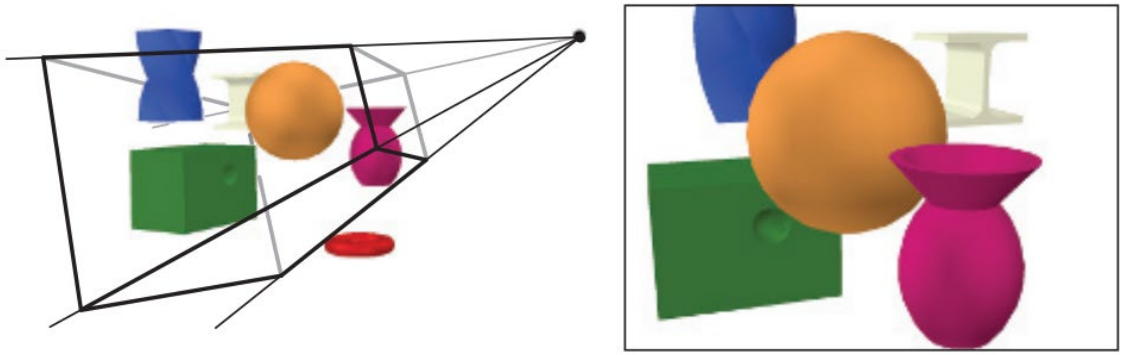
Rendering Pipeline

Chapter 2

The Graphics Rendering Pipeline

"A chain is no stronger than its weakest link."
—Anonymous

This chapter presents the core component of real-time graphics, namely the *graphics rendering pipeline*, also known simply as "the pipeline." The main function of the pipeline is to generate, or *render*, a two-dimensional image, given a virtual camera, three-dimensional objects, light sources, and more. The rendering pipeline is thus the underlying tool for real-time rendering. The process of using the pipeline is depicted in Figure 2.1. The locations and shapes of the objects in the image are determined by their geometry, the characteristics of the environment, and the placement of the camera in that environment. The appearance of the objects is affected by material properties, light sources, textures (images applied to surfaces), and shading equations.



Mathematical models of
Objects +
Camera +
Light +
Screen = Image

Real-Time Rendering, Fourth Edition
Akenine-Moeller, Tomas; Haines, Eric; Hoffman, Naty.

ILLINOIS

Why a Pipeline?

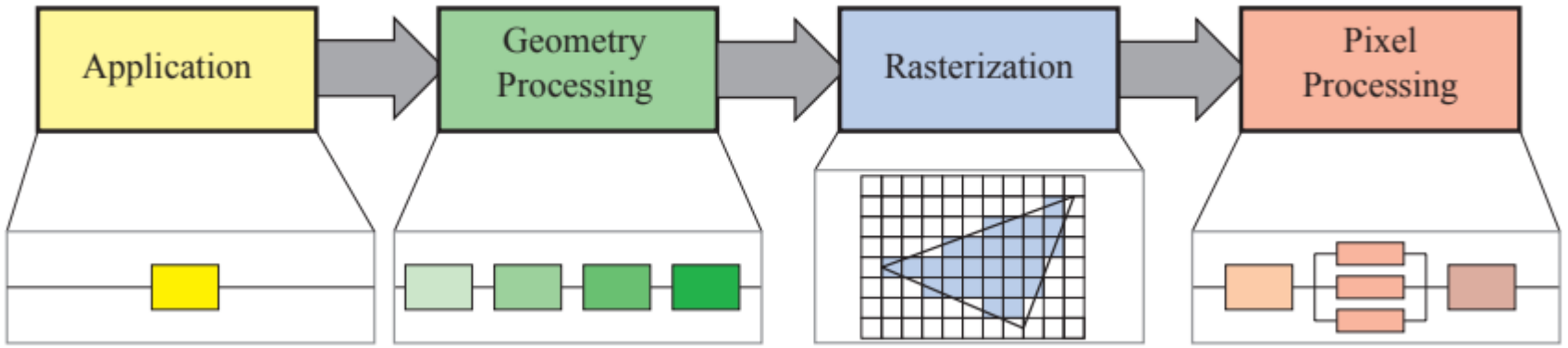
The pipeline stages execute in parallel, with each stage dependent upon the result of the previous stage

Ideally, a non-pipelined system is divided into n pipelined stages could give a speedup of a factor of n

For example, a large number of sandwiches can be prepared quickly by a series of people

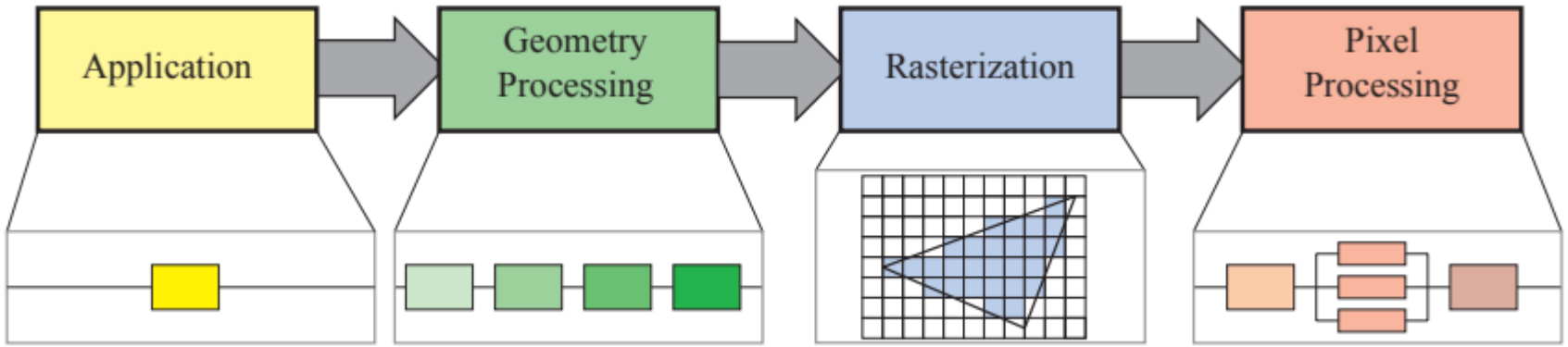
One preparing the bread, another adding meat, another adding toppings

If each person takes twenty seconds for their task, there is a maximum rate of one sandwich every twenty seconds



ILLINOIS

A Rasterization Pipeline



Application tasks: Which objects to render…maybe collision detection

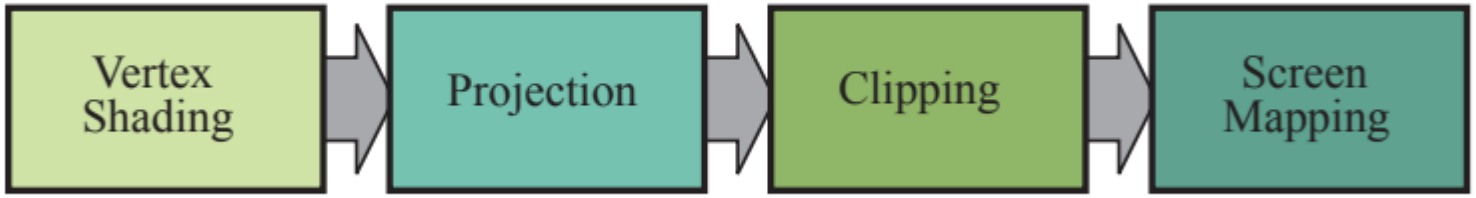
Geometry tasks: Some kinds of animation…projection to a mathematical screen

Rasterization tasks: Which pixels are covered by the projected triangles

Pixel tasks: Shade each pixel based on a model of reflected lights

ILLINOIS

Geometry Processing



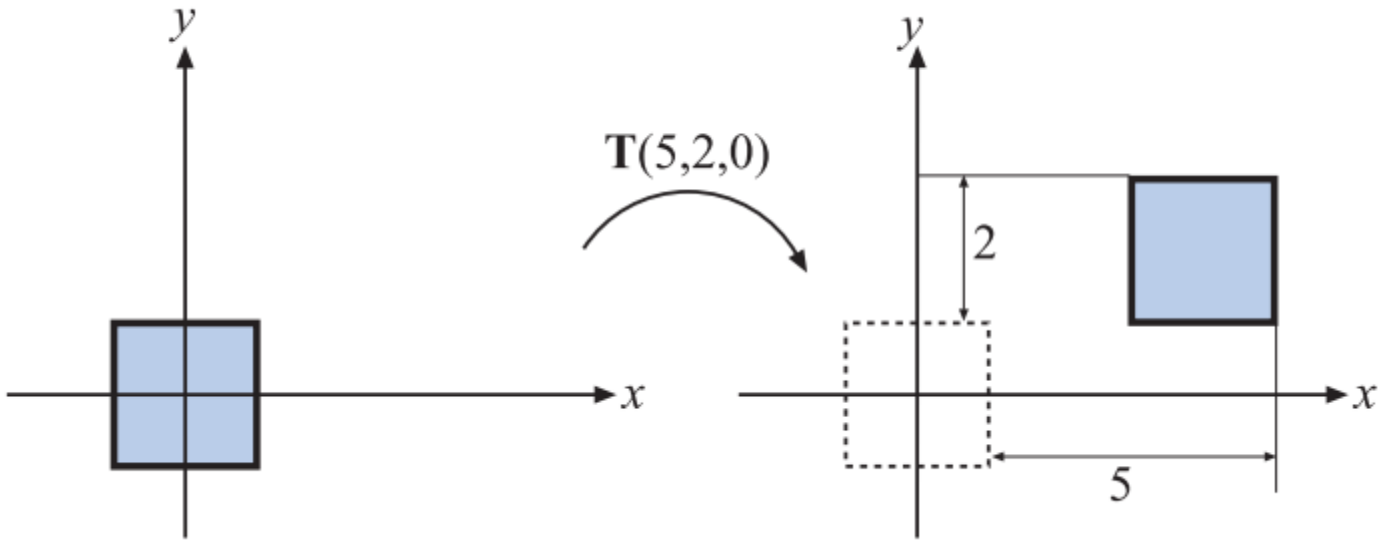
Vertex Shading: Transforming geometry from model to world coordinates

Projection: From 3D onto 2D plane with perspective distortion

Clipping: Discard geometry outside of model screen area

Screen Mapping: From 2D unit square to dimensions of screen in pixels

Vertex Shading: Model Transformation

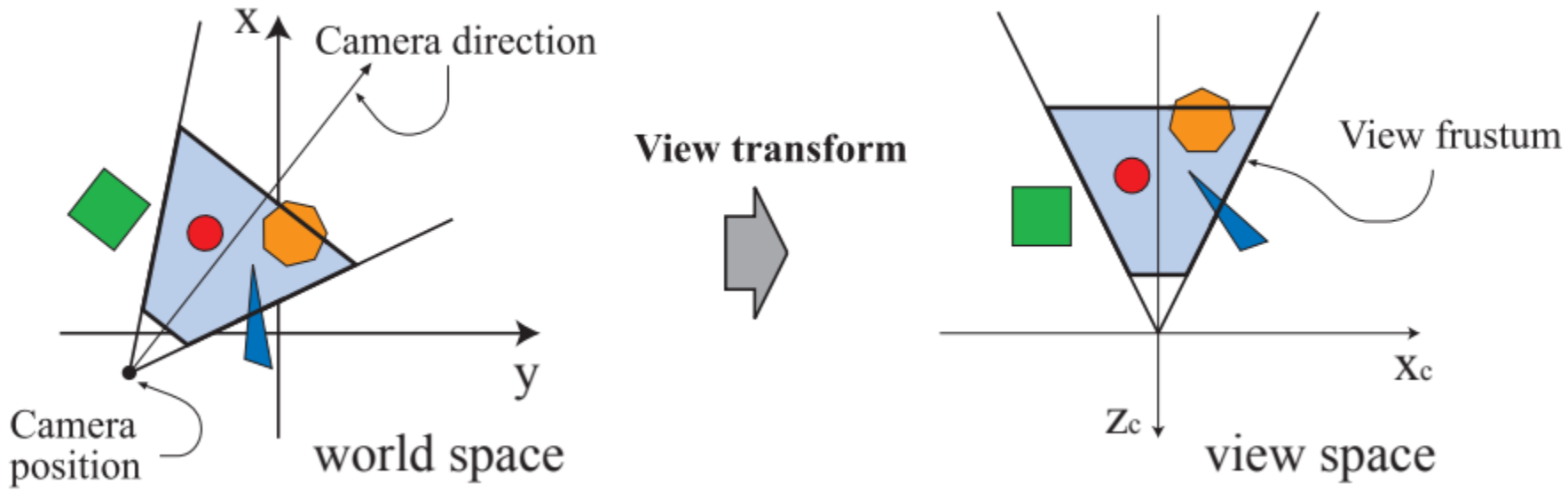


The virtual world of the game has a coordinate system
Meshes are specified in whatever coordinate system the artist used
The Model Transformation transforms coordinates of vertices
So that objects are positioned correctly in the world

ILLINOIS

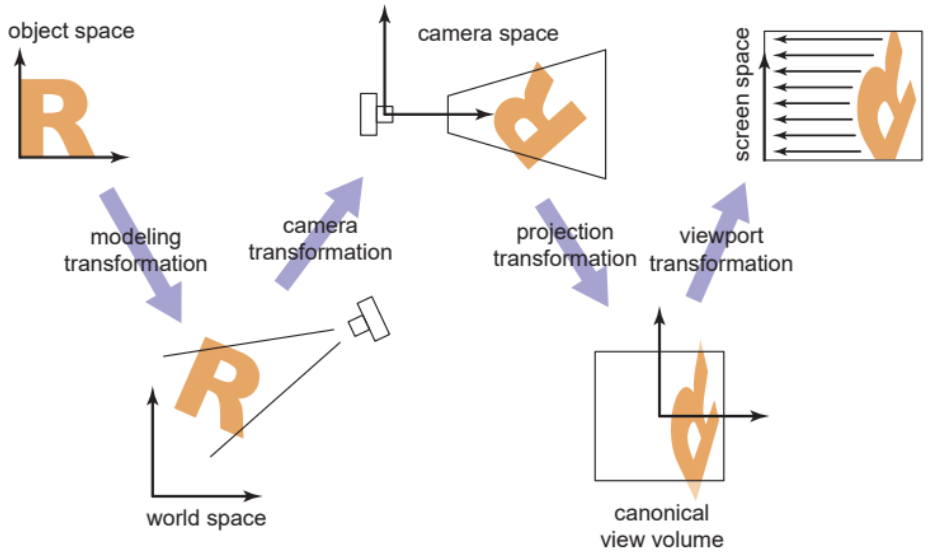
Vertex Shading

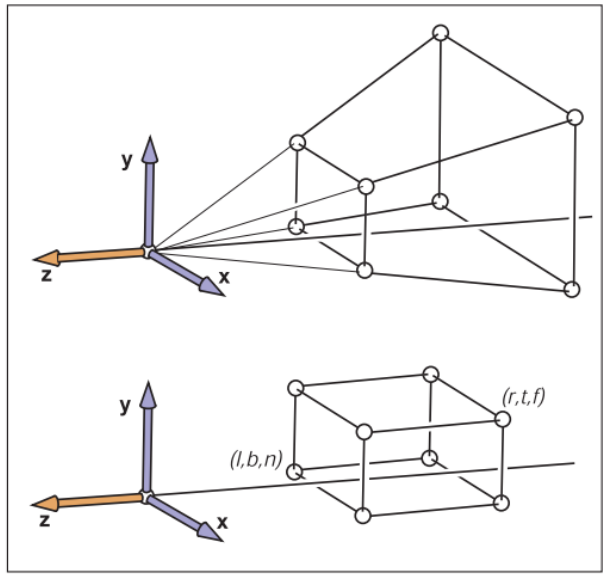
A ***view transformation*** changes the coordinates of the triangle vertices
Camera is at the origin looking down (usually) the z axis with y being up



ILLINOIS

Vertex Shading: Projection

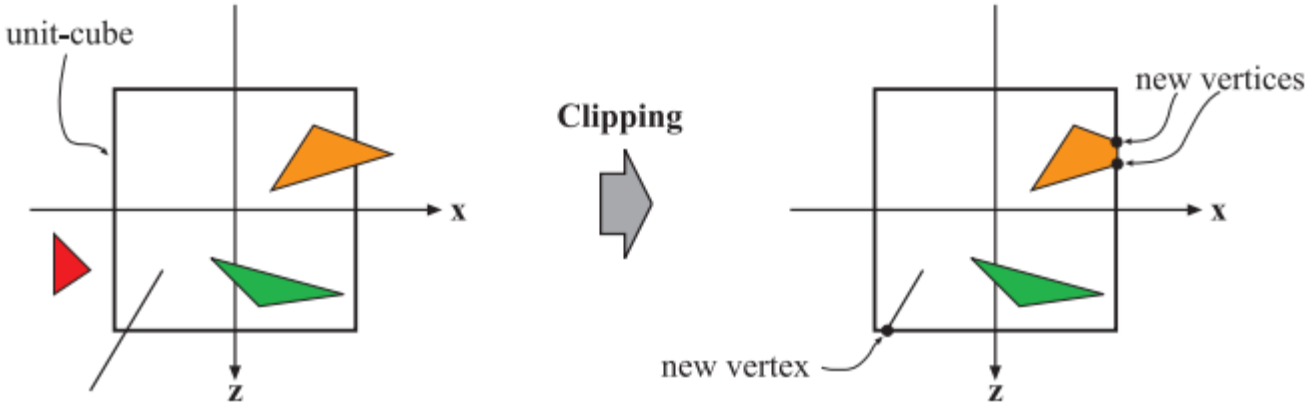




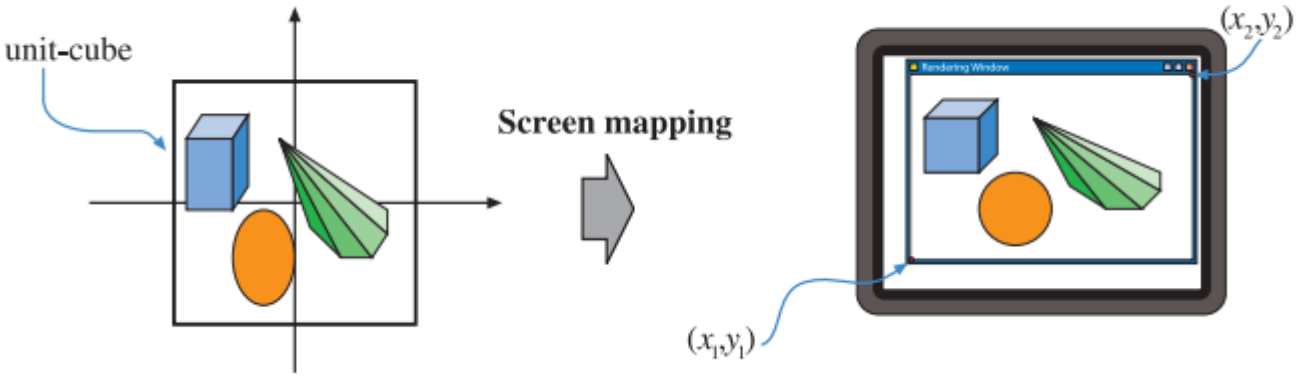
Technically…transforms the geometry so visible volume now in unit cube
(…or a 2x2x2 cube depending on API)
Distorts geometry to look like perspective projection
…can now project to 2D just by disregarding z coordinate
…but that coordinate information is kept around to be used (what for?)

ILLINOIS

Clipping: Get Rid of Geometry that Won't be Seen



Screen Mapping



Coordinates start in range (-0.5,-0.5) to (0.5,0.5)

Range transformed to match the number of pixels in the display window

Just a scaling transformation

ILLINOIS

Rasterization and Pixel Shading

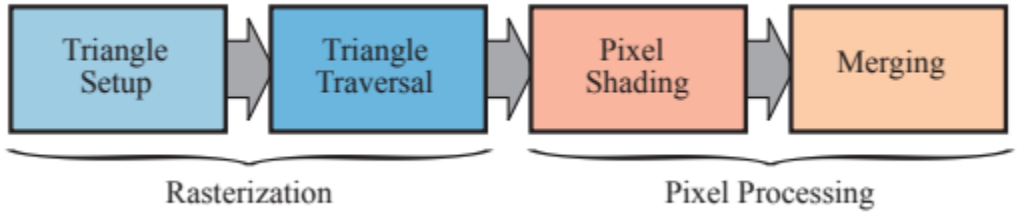


Figure out which vertices belong to which triangles
Compute which discrete pixels are covered by each triangle

Color those pixels (shading)

…then figure out which pixel you generated is closest to the viewer

And that's the end of the pipeline….

ILLINOIS