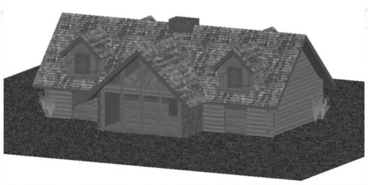
# EECS 672

## Basic Graphics System Concepts

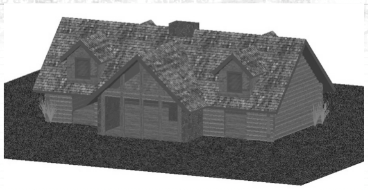
*James R. Miller*
Department of
Electrical Engineering and Computer Science
**The University of Kansas**
Fall 2019



Model
- 3D Geometry
- Attributes (colors, texture, etc.)

Processing
- Line of sight; field of view
- Descriptions of light sources

Image on Display

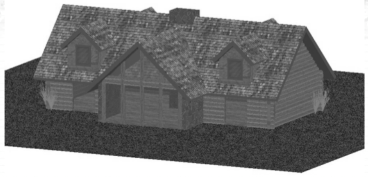


Model
- 3D Geometry
- Attributes (colors, texture, etc.)

Processing
- Line of sight; field of view
- Descriptions of light sources

Image on Display

- Impose convenient reference "model coordinate" (MC) system
- All geometry must be linear (points, lines, triangles)
- Common tool: Piecewise Linear Approximation (PLA)
- Often use nested model coordinate systems

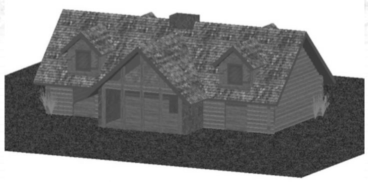


Model
- 3D Geometry
- Attributes (colors, texture, etc.)

Processing
- Line of sight; field of view
- Descriptions of light sources

Image on Display

- Attributes may be "per-vertex" or "per-primitive"
- Coordinate data is simply one type of attribute (typically "per-vertex").
- Attributes can be interpreted in any way in the GLSL program running on the GPU

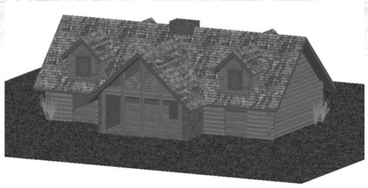


Model
- 3D Geometry
- Attributes (colors, texture, etc.)

Processing
- Line of sight; field of view
- Descriptions of light sources

Image on Display

- Model coordinate (MC) space is infinite in extent
- Ultimately need to map to integer pixels: $0 \le x < xres$ ; $0 \le y < yres$
- 2D applications: scale/translate followed by *clipping*.

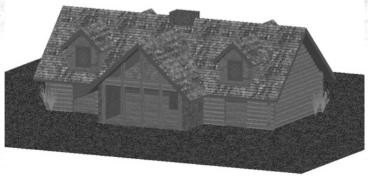


Model
- 3D Geometry
- Attributes (colors, texture, etc.)

Processing
- Line of sight; field of view
- Descriptions of light sources

Image on Display

- Slightly more involved in 3D:
  - Line of sight for orientation (yields "Eye Coordinates" (EC); still infinite extent)
  - 3D➔2D projection & clipping ("projection" subsumes 2D scale/translate)
  - Simulated lighting environment



Model
- 3D Geometry
- Attributes (colors, texture, etc.)

Processing
- Line of sight; field of view
- Descriptions of light sources

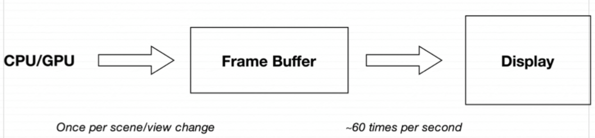
Image on Display

- We need an understanding of the relevant physical aspects of the display.
- How are images produced?
- What is required (e.g., to support animated graphics)?

# Interactive Displays

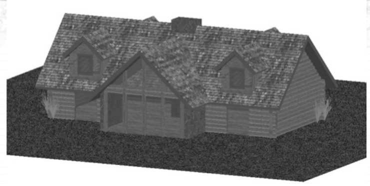
- Many different types
- For our purposes, all have two characteristics in common
  - Physically they are an array of colored dots
  - Once “illuminated”, a dot maintains its “lit color” for a very short time (typically about 1/60 second)
- Cannot require the application to completely reproduce the scene 60 $sec^{-1}$ from the 3D model/view ➔ too much computation for non-trivial scene geometry.
- Instead we use a “Frame Buffer”: a simple low-level representation that permits 60 $sec^{-1}$ refresh with no CPU computation.

# Role of the Frame Buffer



# Frame Buffer

- Frame Buffer is a matrix of digital values
- FrameBuffer[r][c] holds the color for the pixel in row *r*, column *c* of the display window:
  - Color: R, G, B (e.g., one byte each)
- A separate processor redraws the screen 60 $\text{sec}^{-1}$ from this simple low-level representation.
- Optionally one or more of the following can be maintained in parallel when creating a Frame Buffer representation:
  - Alpha (translucency)
  - Depth (distance from observer's eye)
  - Stencil (mask describing what pixels are writeable)
  - …



Model
- 3D Geometry
- Attributes (colors, texture, etc.)

Processing
- Line of sight; field of view
- Descriptions of light sources

Image on Display
- Frame buffer: matrix of colors

- "Processing" yields a Frame Buffer representation of the scene. (CPU-GPU)
- Two issues:
  - Scan conversion (continuous geometry → discrete pixels)
  - Aliasing/anti-aliasing
- Frame buffer represents one "still" image.

# Model-Processing-Image

- The operations discussed for Model-Processing-Image generation were not explicitly assigned to processors (i.e., CPU versus GPU).
- Primary reason: responsibilities can be dynamically distributed. For example, within a single program some pieces of a scene may be more or less completely handled on the CPU, others primarily on the GPU.
- Even within the GPU, operations may be done in different shader programs, based on type of geometry and desired rendering algorithms.
- We will ease our way into these and other possibilities as we progress through the course.

# Animations?

- Animation, simulations, and/or user-controlled view changes need to be perceived as being "smooth".
- Each frame of an animated sequence must be generated by (i) clearing the frame buffer, and (ii) redrawing the scene with updated model and view specifications.
- When using a single frame buffer, there will usually be a noticeable "flashing" between frames.
- "Double buffering" eliminates this problem and allows smooth motion.

# What's Next?

- With this brief background, we will begin our study of graphics using OpenGL by examining a series of example programs that can be accessed from:

*http://people.eecs.ku.edu/~jrmiller/Courses/OpenGL/OpenGL.html*

*Current OpenGL versions on EECS Workstations*:

```
VERSIONS: GL: 4.5.0 NVIDIA 384.130
        GLSL: 4.50 NVIDIA
        GLFW: 3.1.2 X11 GLX clock_gettime /dev/js XI Xf86vm shared
```