Lecture 15:
Perspective Projection Pipeline

October 22, 2019

How do Pixels get Colored?

- Step back and consider two alternatives

```
for (each_object)
    render (object)
```

- Reasoning goes “What pixels are altered by this object?”
- Basis for OpenGL rendering pipeline.
- Highly Parallel on modern hardware.
- Fast, but object interactions extremely hard.

```
for (each_pixel)
    color (pixel)
```

- Reasoning goes “What object alter this pixels coloring?”
- Basis for Ray Casting and Ray Tracing.
- Highly Parallel
- Cost mainly intersecting 3D objects with light rays cast back into the scene.

Fill Pixels Inside Triangles

AD-761 965

HALF-TONE PERSPECTIVE DRAWINGS BY COMPUTER

Chris Wylie, et al

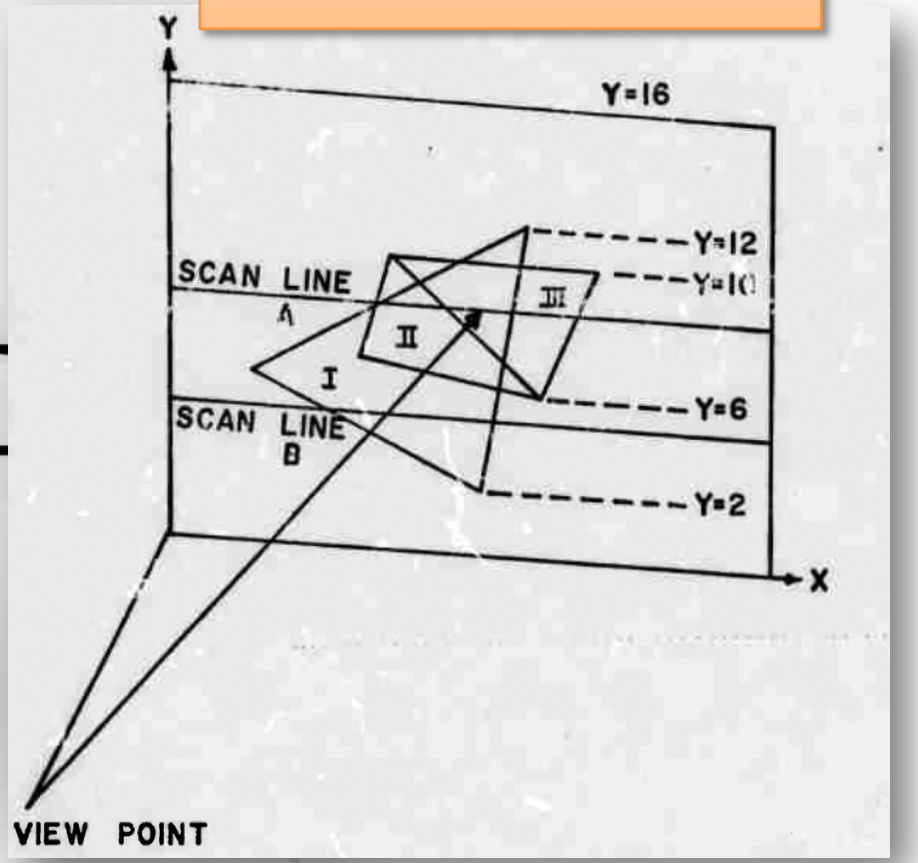
Utah University

Prepared for:

Advanced Research Projects Agency

12 February 1968

Fifty years ago
"Scan Conversion"



A Characterization of Ten Hidden-Surface Algorithms

Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker
ACM Computer. Surveys. 6, 1 (March 1974)

A Characterization of Ten Hidden-Surface Algorithm

IVAN E. SUTHERLAND*, ROBERT F. SPROULL**, AND ROBERT A. SCHUMACKER

This paper discusses the hidden-surface problem from the point of view of sorting. The various surfaces of an object to be shown in hidden-surface or hidden-line form must be sorted to find out which ones are visible at various places on the screen. Surfaces may be sorted by lateral position in picture (XY), by depth (Z), or by other criteria. The paper shows that the order of sorting and the types of sorting used form differences among the existing hidden-surface algorithms To reduce the work of sorting, each algorithm capitalizes on some coherence property of the objects represent "Scan-line coherence," the fact that one TV scan line of output is likely to be nearly the same as the previous TV scan line, is one commonly used kind of coherence. "Frame coherence," the fact that the entire picture doe not change very much between successive frames of a motion picture can be very helpful if it is applicable.

Forty six years ago, draw triangles so closest occludes all others.

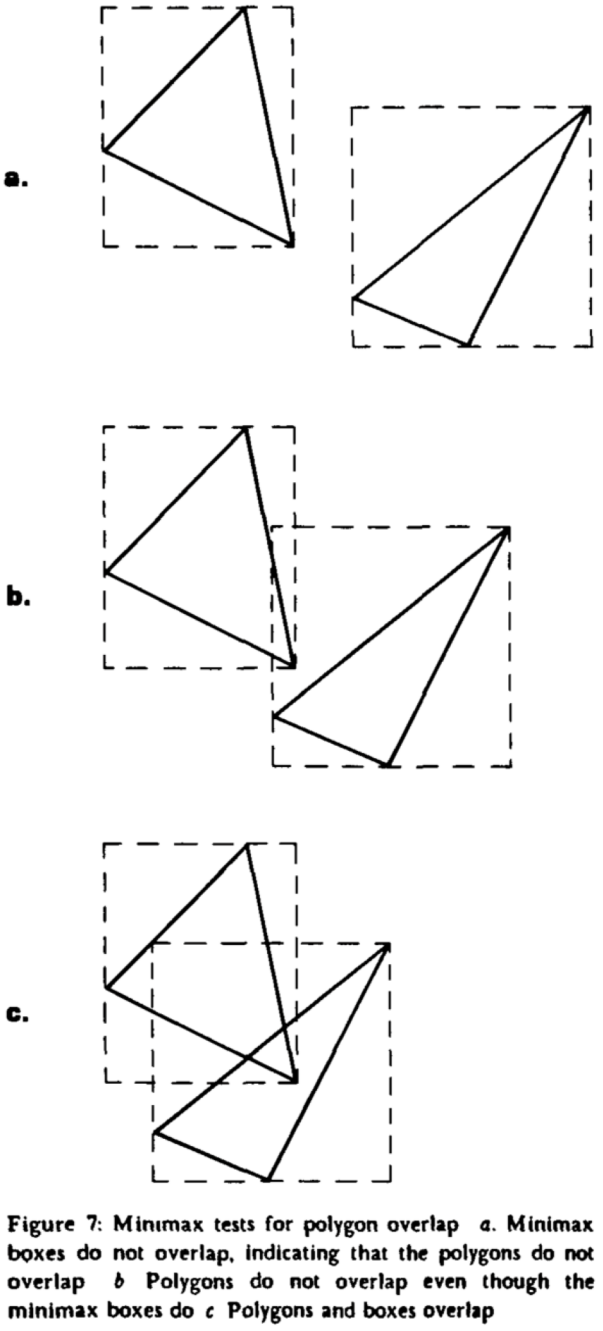


Figure 7: Minimax tests for polygon overlap *a*. Minimax boxes do not overlap, indicating that the polygons do not overlap *b* Polygons do not overlap even though the minimax boxes do *c* Polygons and boxes overlap

Computer Graphics Volume 16, Number 3 July 1982

The Geometry Engine: A VLSI Geometry System for Graphics

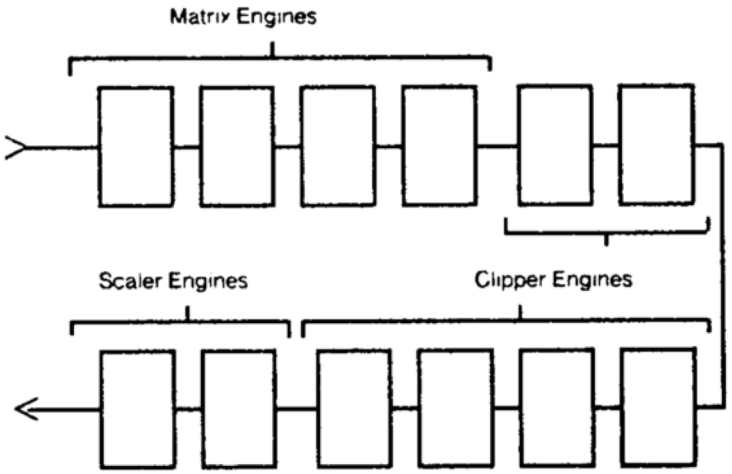
by

James H. Clark

Computer Systems Laboratory
Stanford University
and
Silicon Graphics, Inc.
Palo Alto, California

Geometry System Functions

The Geometry System [2] is designed for high-performance, low-cost, floating-point geometric computation in computer graphics applications. It is composed of three subsystems, each of which is composed of Geometry Engines. These subsystems are illustrated in Figure 3. The particular position of a Geometry Engine in the pipeline determines its particuar function in the whole system. Each Engine has a configuration register that is loaded when the system is powered on, after a Reset command is issued. Until the system is reset again, the Engine behaves according to the configuration code.



In Essence: The **Perspective Projection Pipeline**

The subsystems are:

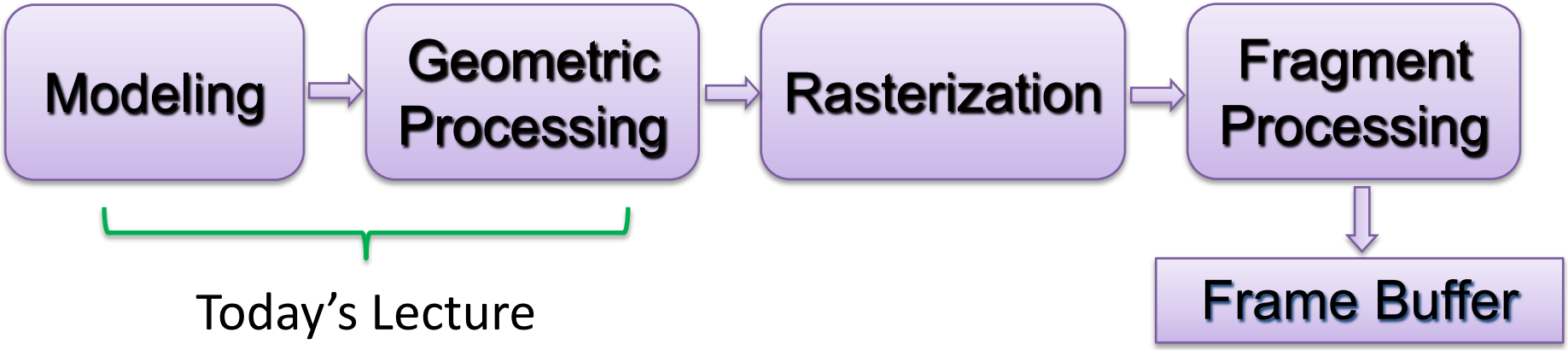
- **Matrix Subsystem** - A stack of 4x4 floating-point matrices for completely general, 2D or 3D floating-point coordinate transformation of graphical data.
- **Clipping Subsystem** - A windowing, or clipping, capability for clipping 2D or 3D graphical data to a window into the user's virtual drawing space. In 3D, this window is a volume of the user's virtual, floating-point space, corresponding to a truncated viewing pyramid with "near" and "far" clipping.
- **Scaling Subsystem** - Scaling of 2D and 3D coordinates to the coordinate system of the particular output device of the user. In 3D, this scaling phase also includes either orthographic or perspective projection onto the viewer's virtual window. Stereo coordinates are computed and optionally supplied as the output of the system.

The characteristics of each of these subsystems follows.

Thirty six years ago, the foundation of Silicon Graphics and beginning of OpenGL.

Four Major Tasks

- Think of rending objects in terms of:
  - Modeling
  - Geometry Processing
  - Rasterization
  - Fragment Processing



Moving to Formulation #3

- Review, #1 origin at PRP.
- Review, #2 origin at image center.
- Review, #1 and #2
  - No useful information on the z-axis
- We now have a new goal
- Project into a canonical view volumne
- A rectangular volume with bounds:
  - U: -1 to 1, V: -1 to 1, D: 0 to 1

Remember When We Started

- What happens if you multiply a point in homogeneous coordinates by a scalar?
- Nothing!

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} sx \\ sy \\ sz \\ sw \end{bmatrix} = s \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Scalar Multiplication Continued

- Multiply a homogeneous matrix by a scalar?
- Again, nothing changes.

$$\begin{bmatrix} ax+by+cz+dw \\ ex+fy+gz+hw \\ ix+jy+kz+lw \\ mx+ny+oz+pw \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$\begin{bmatrix} ax+by+cz+dw \\ ex+fy+gz+hw \\ ix+jy+kz+lw \\ mx+ny+oz+pw \end{bmatrix} = \begin{bmatrix} s(ax+by+cz+dw) \\ s(ex+fy+gz+hw) \\ s(ix+jy+kz+lw) \\ s(mx+ny+oz+pw) \end{bmatrix} = s \cdot \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Form #1 & Textbook Derivation

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{vmatrix}$$

Lecture Form #1

Equivalent

$$\begin{vmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix}$$

Chapter 7 Viewing

7.3 Perspective Projection

The mechanism of projective transformations makes it simple to implement the division by z required to implement perspective. In the 2D example shown in Figure 7.8, we can implement the perspective projection with a matrix transformation as follows:

$$\begin{bmatrix} y_s \\ 1 \end{bmatrix} \sim \begin{bmatrix} d & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix}.$$

This transforms the 2D homogeneous vector $[y; z; 1]^T$ to the 1D homogeneous vector $[dy\ z]^T$,

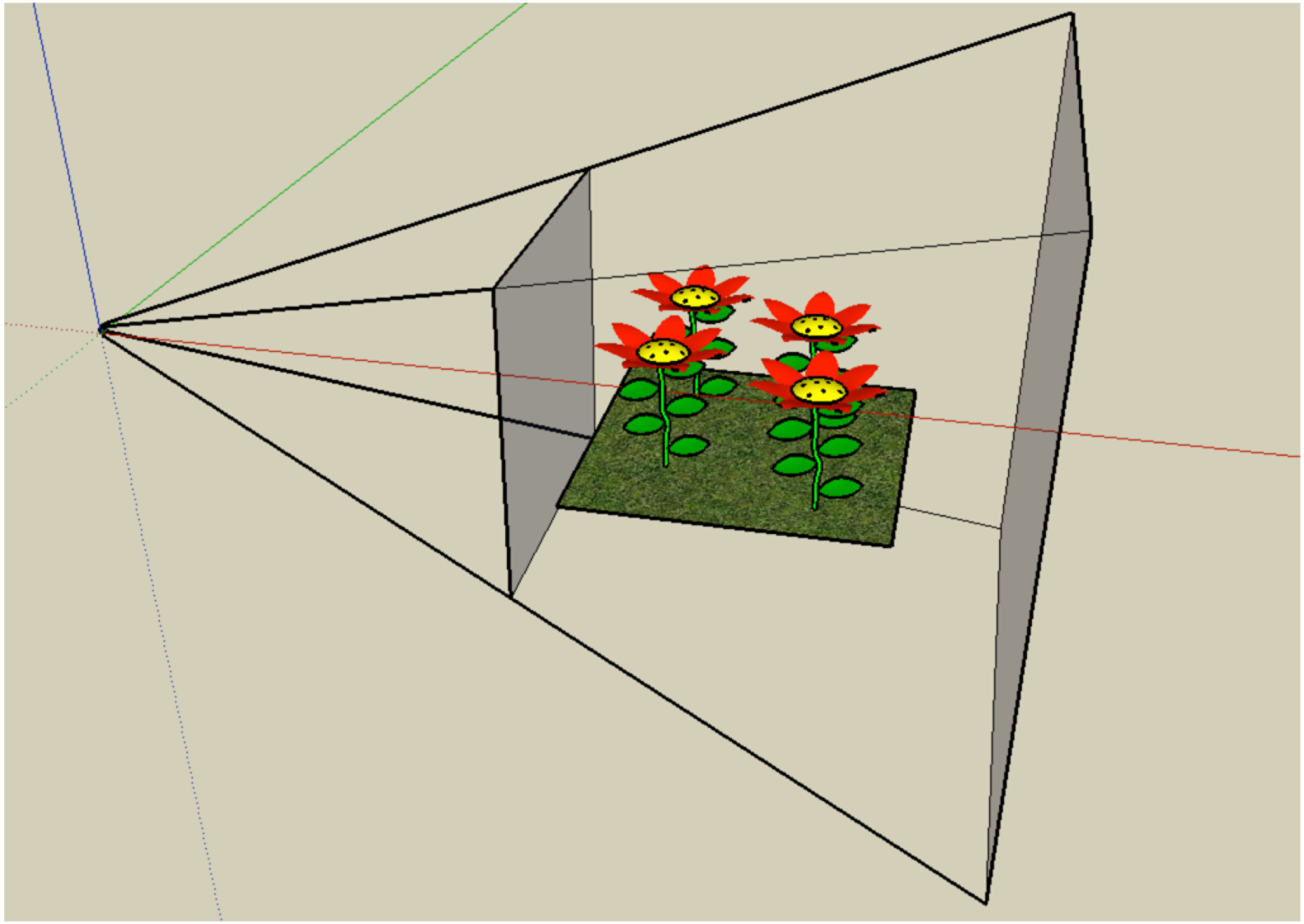
Now introduce clipping planes

- Text introduces *n*, the near clipping plane.
- Also introduces *f*, the far clipping plane.
- Sets what first called *d* to *n*.
- The handling of *z* now carries information?

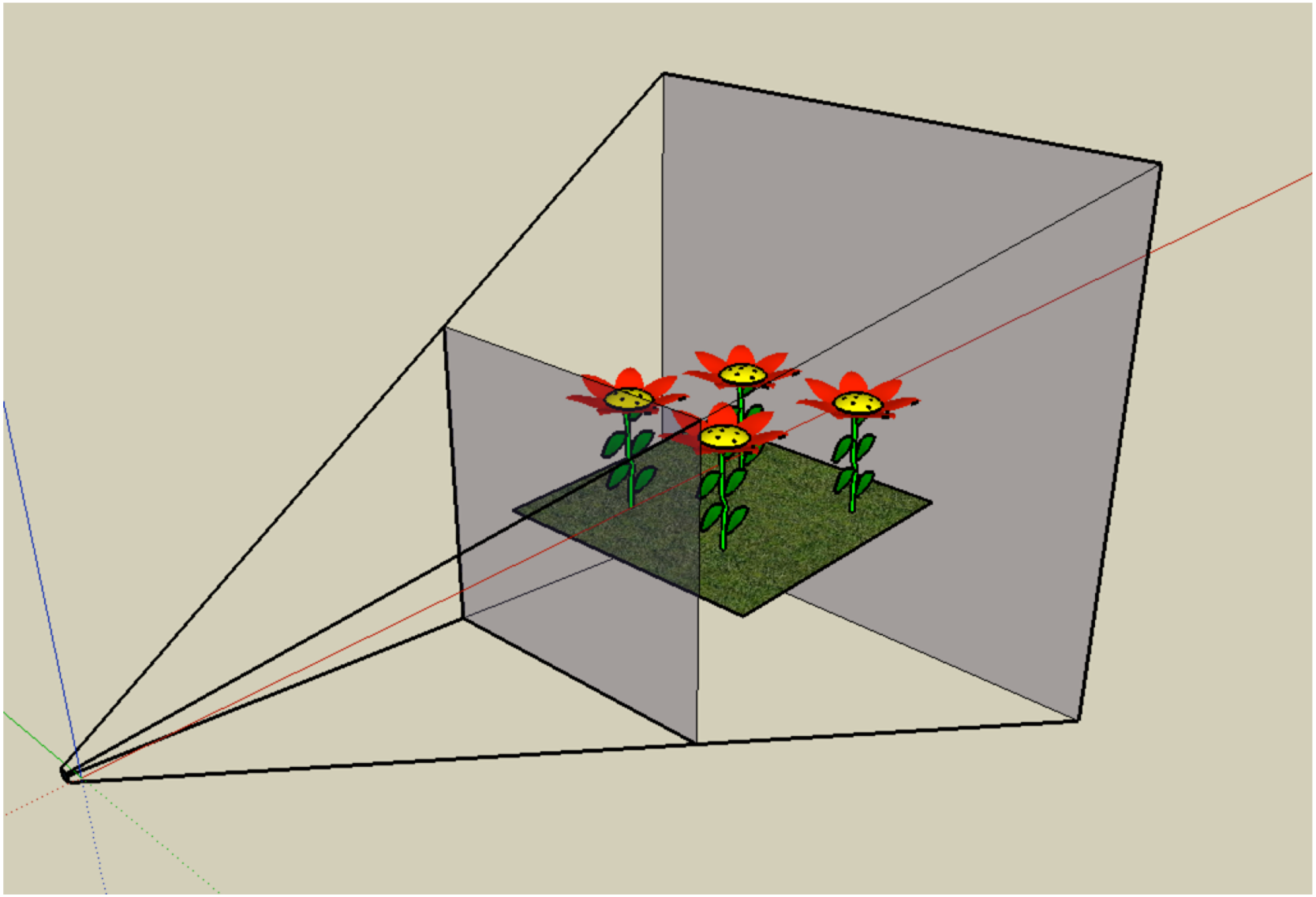
$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} nx \\ ny \\ zn+zf-fn \\ z \end{bmatrix} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

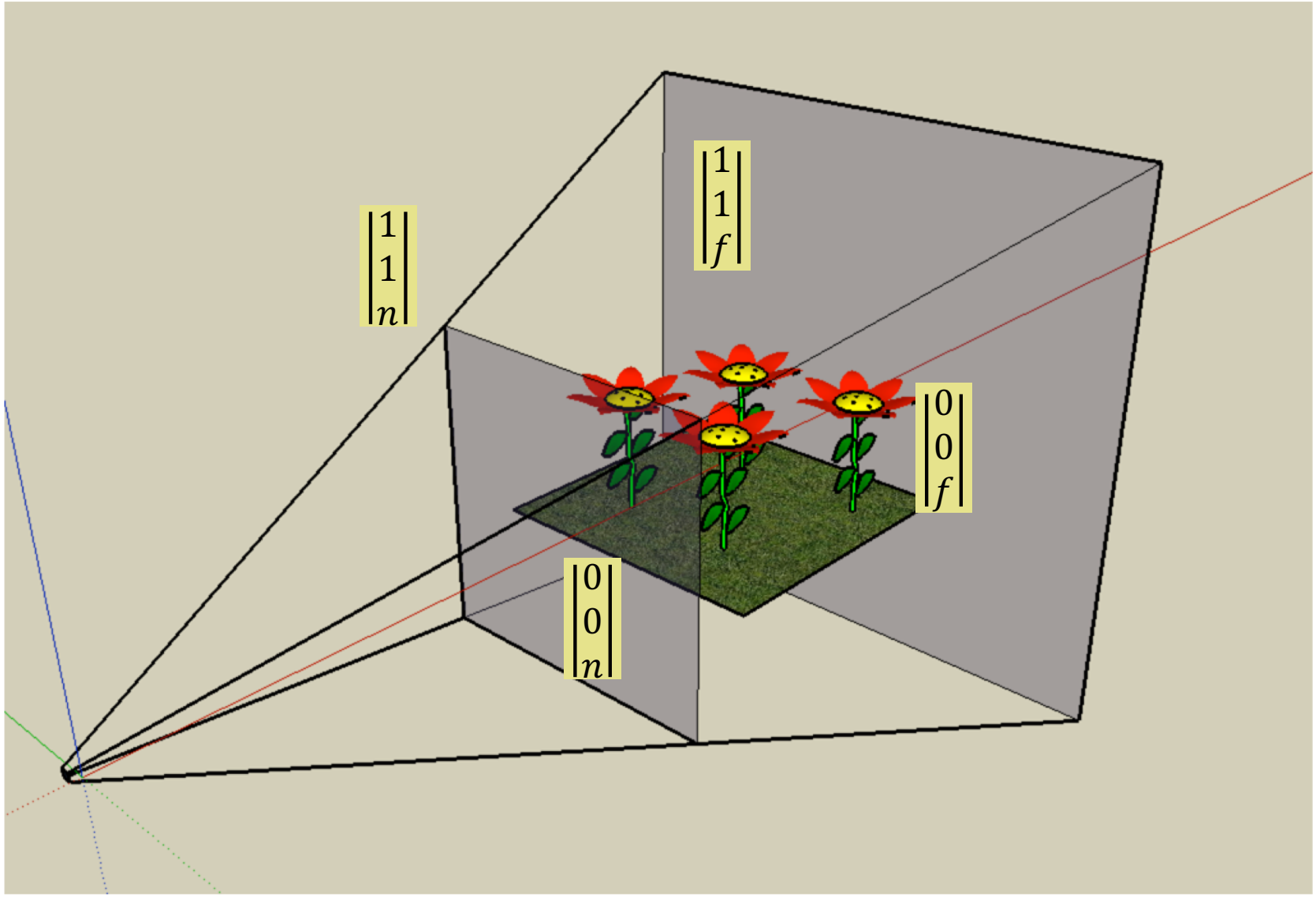
Visualize View Volume (View 1)



Visualize View Volume (View 2)



Consider Some Key Points



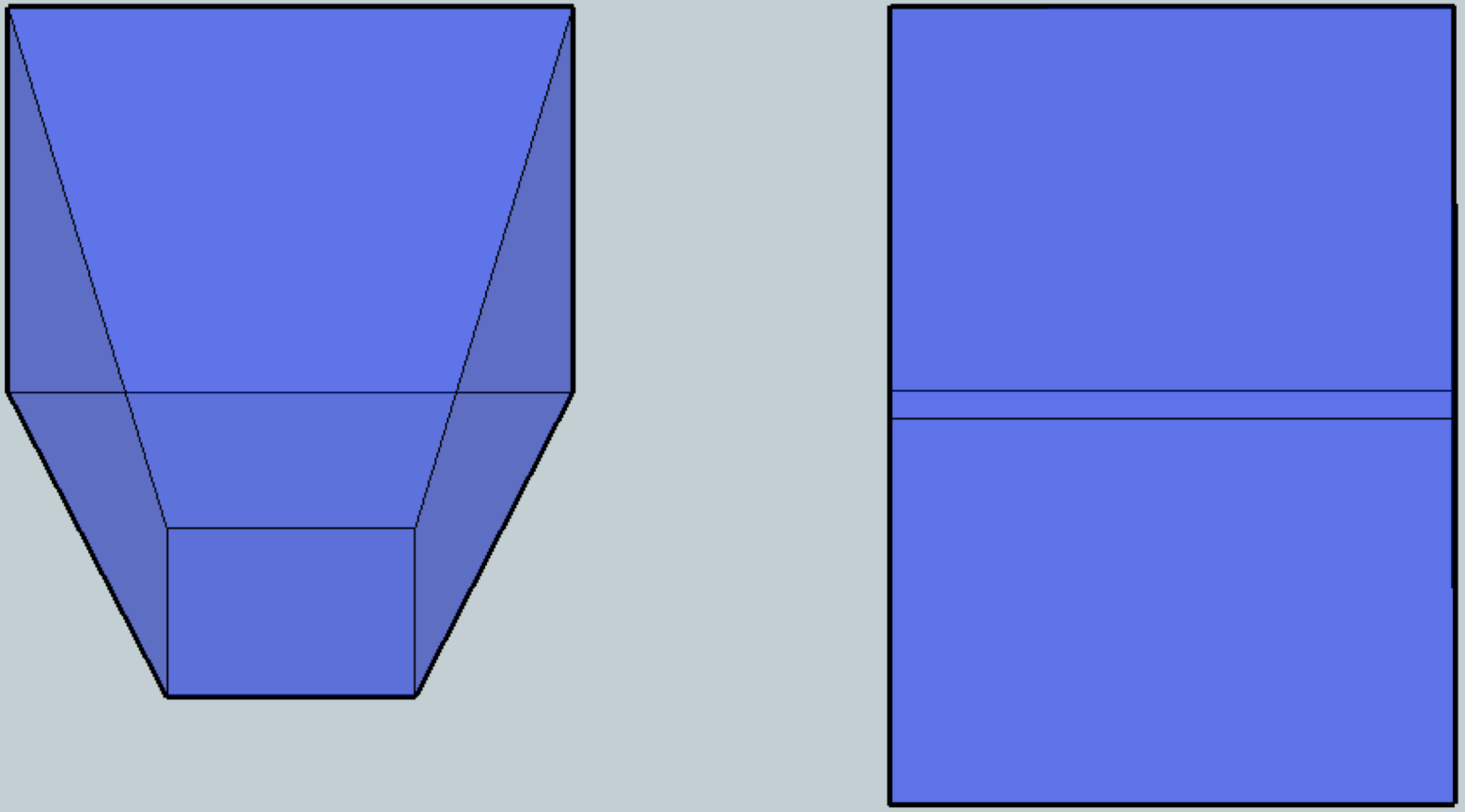
Formulation #3: The Algebra

$$\begin{pmatrix} 0 & 0 & 1 & \frac{n}{f} \\ 0 & 0 & 1 & \frac{n}{f} \\ \frac{(f+n)n-fn}{n} & \frac{(f+n)f-fn}{f} & \frac{(f+n)n-fn}{n} & \frac{(f+n)f-fn}{f} \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & f+n & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ n & f & n & f \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 1 & \frac{n}{f} \\ 0 & 0 & 1 & \frac{n}{f} \\ n & f & n & f \\ 1 & 1 & 1 & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & f+n & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{M_P} * \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ n & f & n & f \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

The 1,1 corner of the near clipping plane maps to 1,1 on image plane. The 1,1 corner on the far clipping plane comes toward the optical axis and becomes *n/f* on the image plane.

Visualize Frustum Change



SageMath Notebook CS410lec17n01

3D Projective Viewing with House

Ross Beveridge, October 23, 2018

It is helpful to be able to review the linear algebra and the 3D geometry of viewing using a tool such as Sage. The example setup here originated as a series of Maple Worksheets I developed in CS 410 nearly 20 years ago. Here the example is updated and this example carries two key ideas. You should see the mathematics of the four major components of the Perspective Projection Pipeline geometry in both symbolic form and also in specific numerical examples. Second, the object moddel, a house, can be viewed using a 3D viewer both before and after being transformed in a camera's canoncial view volume.

Keep in mind the canonical view volume is a cube bounded between -1 and 1 along the camera's X, Y and Z axes. The X and Y axes are the horizontal and vertical axes of the image plane. The Z axes carries pseudo-depth information. The 'psuedo' is appened to the front of 'depth' to highlight that while larger values mean further from the camera, a non-linear warping is introduced. This can be seen in the examples below.

This one Notebook includes both symbolic forms of the key components in the transformation along with different concrete examples. There are a series of different examples supported in this worksheet. These are defined and selected between below. First, some basic Sage graphics defaults are customized and then the house model is defined and viewed.

Geometry Pipeline

$$M_{WC} = M_O\, M_P\, M_R\, M_T$$

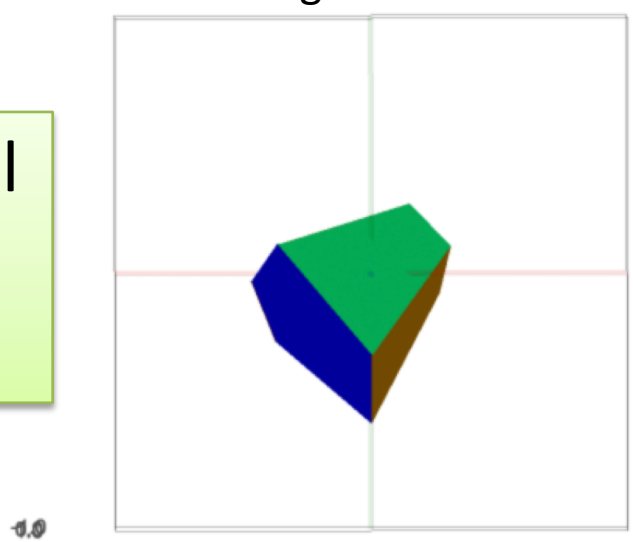
Understand each of these!

SageMath Notebook CS410lec15n01

$$M_{WC} = \begin{pmatrix} -\frac{2}{13}\sqrt{17}\sqrt{\frac{13}{17}} & 0 & \frac{4}{39}\sqrt{17}\sqrt{\frac{13}{17}} & -\frac{40}{13}\sqrt{17}\sqrt{\frac{13}{17}} \\ \frac{8}{39}\sqrt{\frac{13}{17}} & -\frac{2}{3}\sqrt{\frac{13}{17}} & \frac{4}{13}\sqrt{\frac{13}{17}} & -\frac{120}{13}\sqrt{\frac{13}{17}} \\ -\frac{22}{153}\sqrt{17} & -\frac{22}{153}\sqrt{17} & -\frac{11}{51}\sqrt{17} & \frac{2860}{153}\sqrt{17} - \frac{200}{9} \\ \frac{2}{17}\sqrt{17} & \frac{2}{17}\sqrt{17} & \frac{3}{17}\sqrt{17} & -\frac{260}{17}\sqrt{17} \end{pmatrix}$$

Canonical View Volume

Image Plane



Looking Down