Rendering

Scene Description

3D objects
Coordinate Frame
Camera(s)
Materials
Lights

?  

2D Image
OpenGL Rendering Pipeline
(with some details abstracted away)

Javascript, three.js

GLSL program
vertex shader

“fixed function”
rasterization

GLSL program
fragment shader

Framebuffer
Thus far...

- triangles
- vertices: $v = [x \ y \ z]^T$ local coords, on GPU
- vertex shader: $v' = M \ v$ to image coords
- fragment shader: colour = (1,0,0) 
  colour = $N \cdot L$
- instancing: redraw with $M_1, M_2, M_3, \ldots$
- many coordinate frames:
  e.g., wheel $\rightarrow$ car $\rightarrow$ world $\rightarrow$ camera $\rightarrow$ image
Algorithm: “Projective Rendering”

for each frame
  clear screen
for each object instance
  for each triangle j   // project onto image:
    transform vertices   // vertex shader
    for each pixel in j  // rasterization
      compute colour    // fragment shader
Linear Algebra Review

vectors

dot product
Math Review

*matrix-vector multiplication*

(a) as dot products with the rows

(b) as weighted combinations of the columns
Math Review

Cross Product

Right Handed Coordinate System

(curl fingers from $u$ to $v$; thumb points to $u \times v$)
Math Review

Coordinate Systems

Right-handed Coordinate System

Left-handed Coordinate System

using right-hand rule

using left-hand rule
**Math Review**

*Points and Vectors*

**affine space:**
allows vector-to-point addition

**vector space**
vectors are invariant under translation
Math Review

Coordinate System vs Frame

coordinate system:
frame:
Math Review

**Working with Frames**

\[ P = O + x\hat{i} + y\hat{j} \]
Many Coordinate Frames in a Scene
(and using transformation matrices to move between them)