Learning Objectives

- Students completing this lecture will be able to
  - Describe the differences between parallel projection and perspective projection
  - Explain the following terms: multiview orthographic projection, vanishing point
  - Distinguish isometric, dimetric, and trimetric projections; one-point, two-point, and three-point perspectives
  - Write WebGL code to set up computer viewing

Classical Viewing

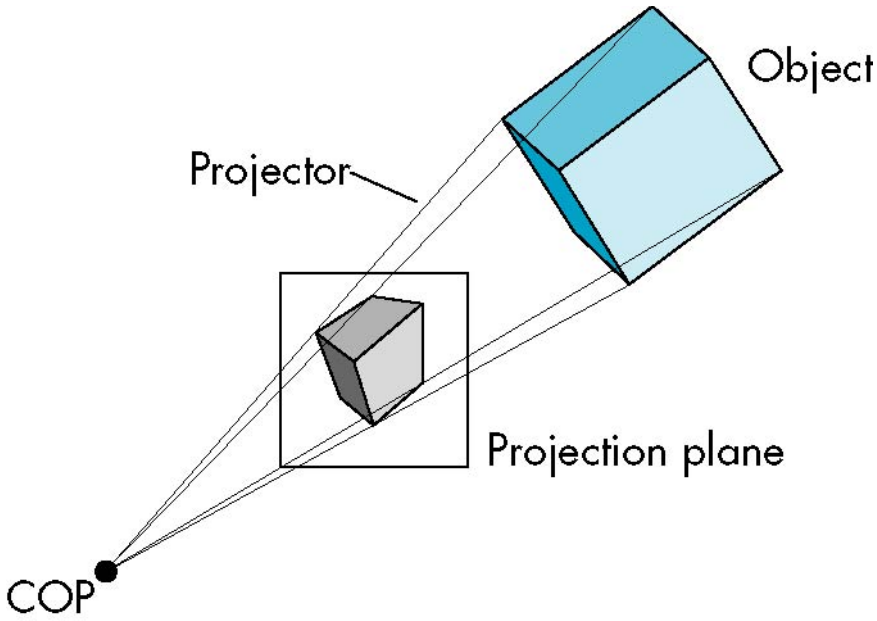
Classical Viewing

- Viewing requires three basic elements
  - One or more objects
  - A viewer with a projection surface
  - Projectors that go from the object(s) to the projection surface
- Classical views are based on the relationship among these elements
  - The viewer picks up the object and orients it how she would like to see it
- Each object is assumed to constructed from flat principal faces
  - Buildings, polyhedra, manufactured objects

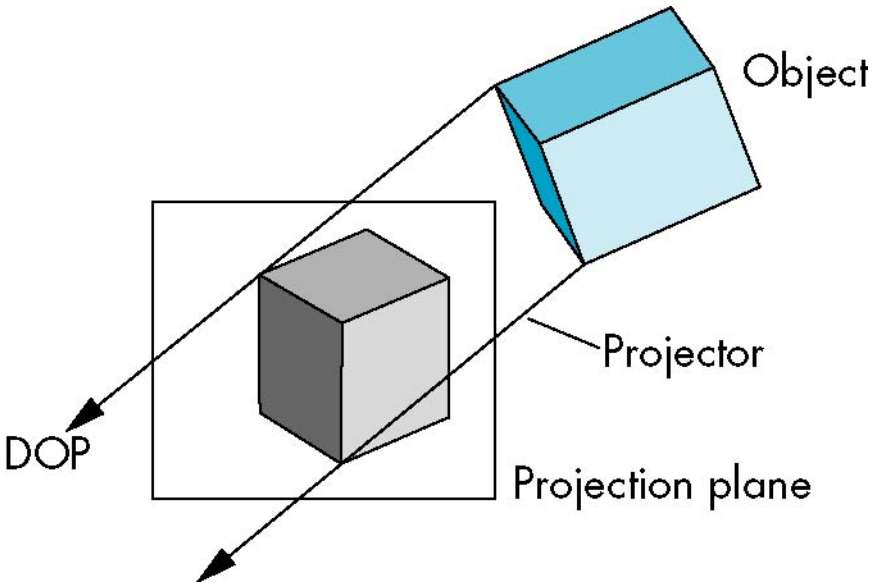
Perspective vs. Parallel

- Computer graphics treats all projections the same and implements them with a single pipeline
- Classical viewing developed different techniques for drawing each type of projection
- Fundamental distinction is between parallel and perspective viewing even though mathematically parallel viewing is the limit of perspective viewing (i.e., moving the center of projection to the infinity)

Perspective Projection vs. Parallel Projection

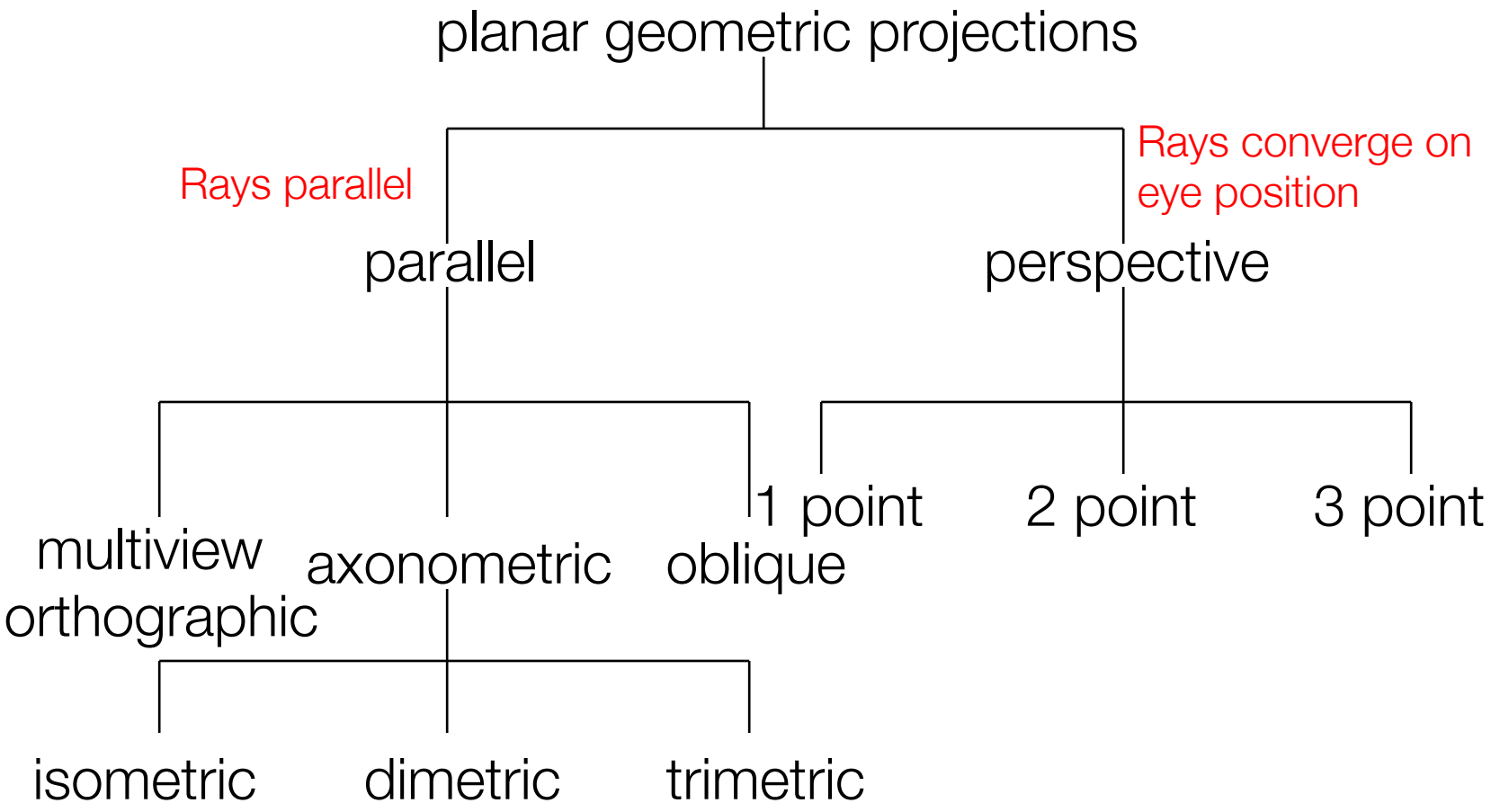


Center of projection

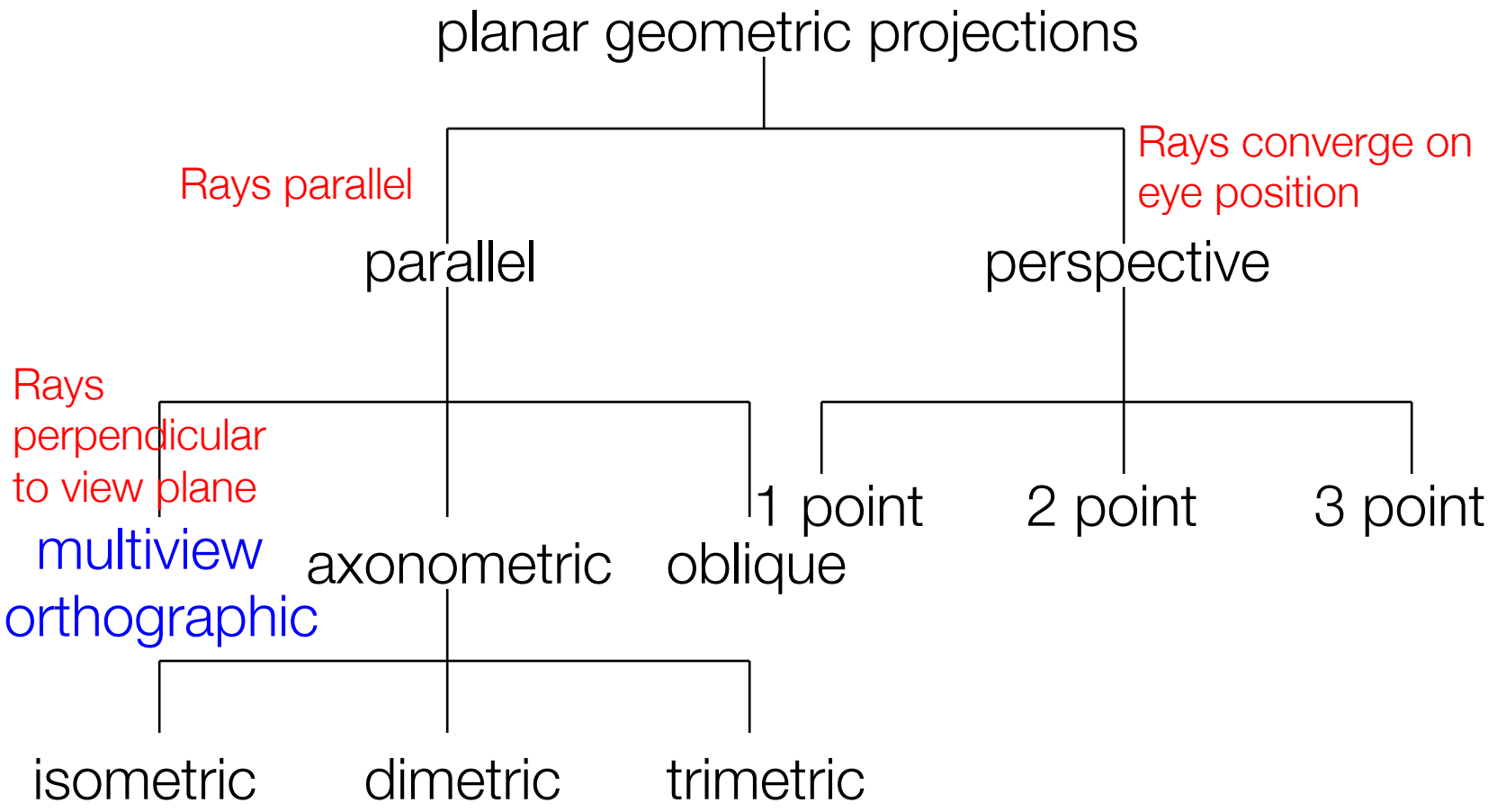


Direction of projection

Taxonomy of Planar Geometric Projections

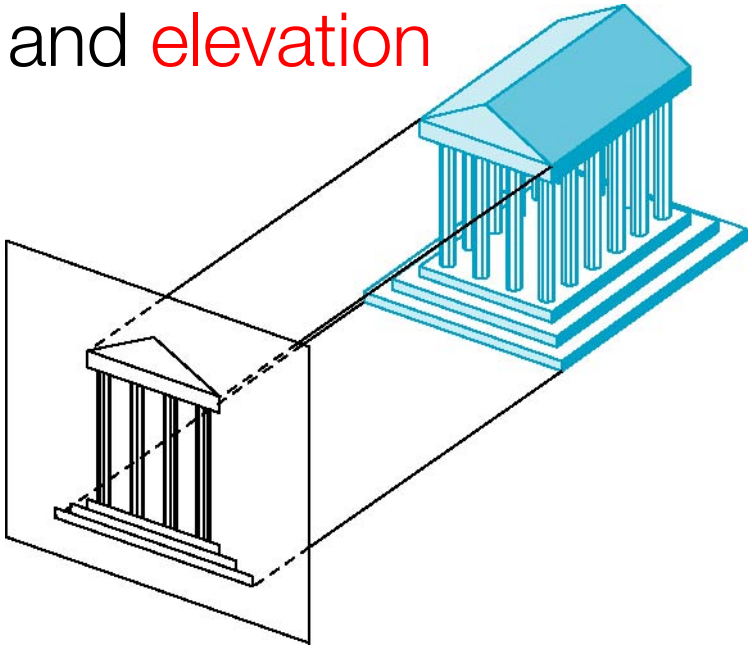


Taxonomy of Planar Geometric Projections



Orthographic Projection

- Orthographic projections are a small set of transforms often used to show profile, detail or precise measurements of a three dimensional object
- Projectors are orthogonal to projection surface
- Common names for orthographic projections include plane, cross-section, bird's-eye, and elevation



Multiview Orthographic Projection

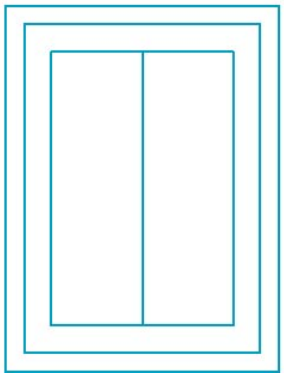
- Projection plane parallel to principal face
- Usually form front, top, side views

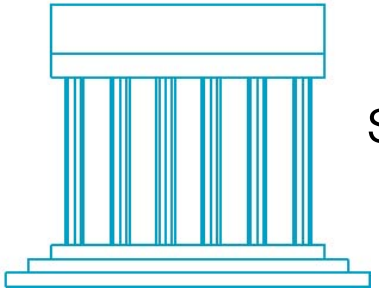
isometric (not multiview orthographic view)

front

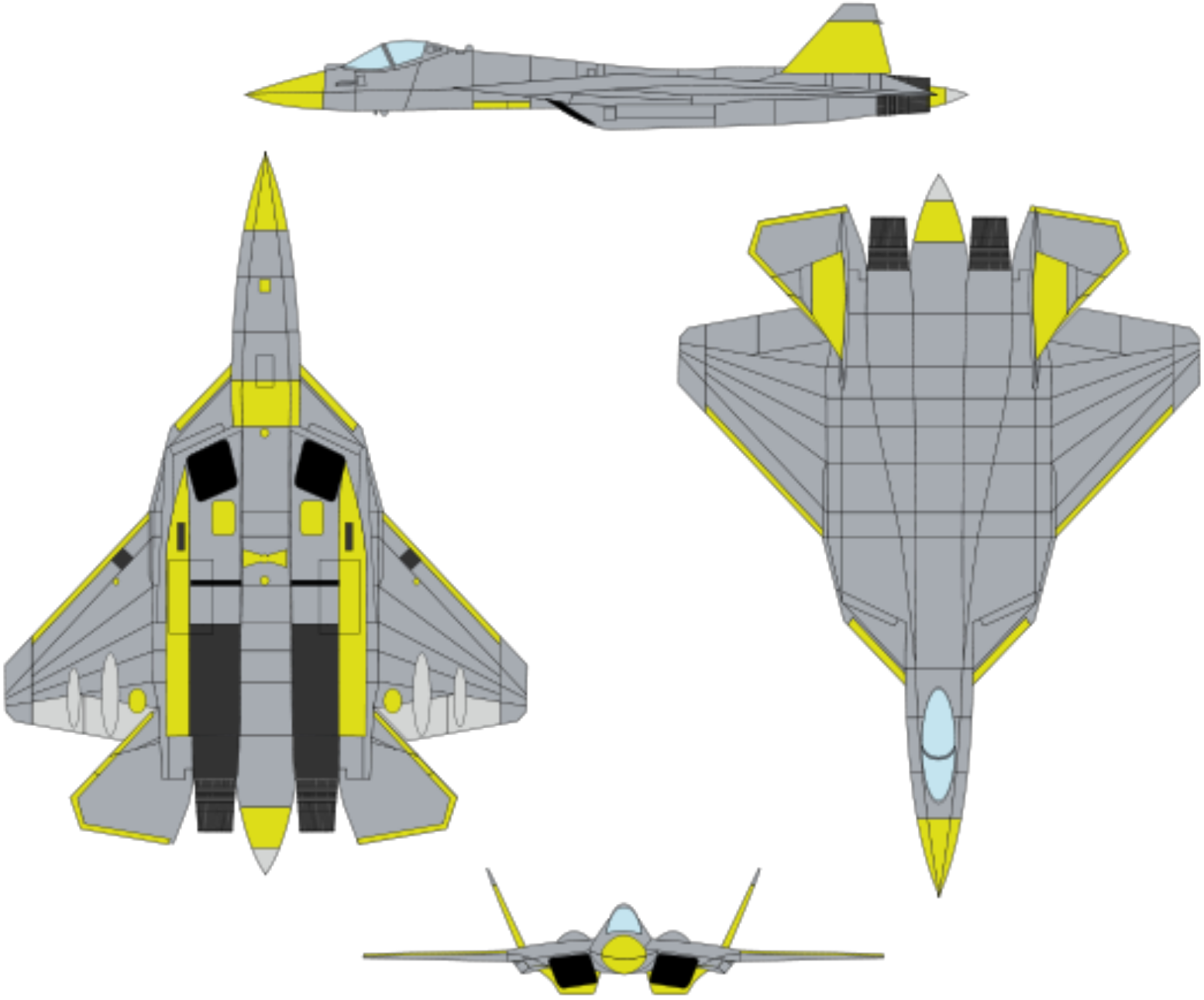
In CAD and architecture, we often display three multiviews plus isometric

top





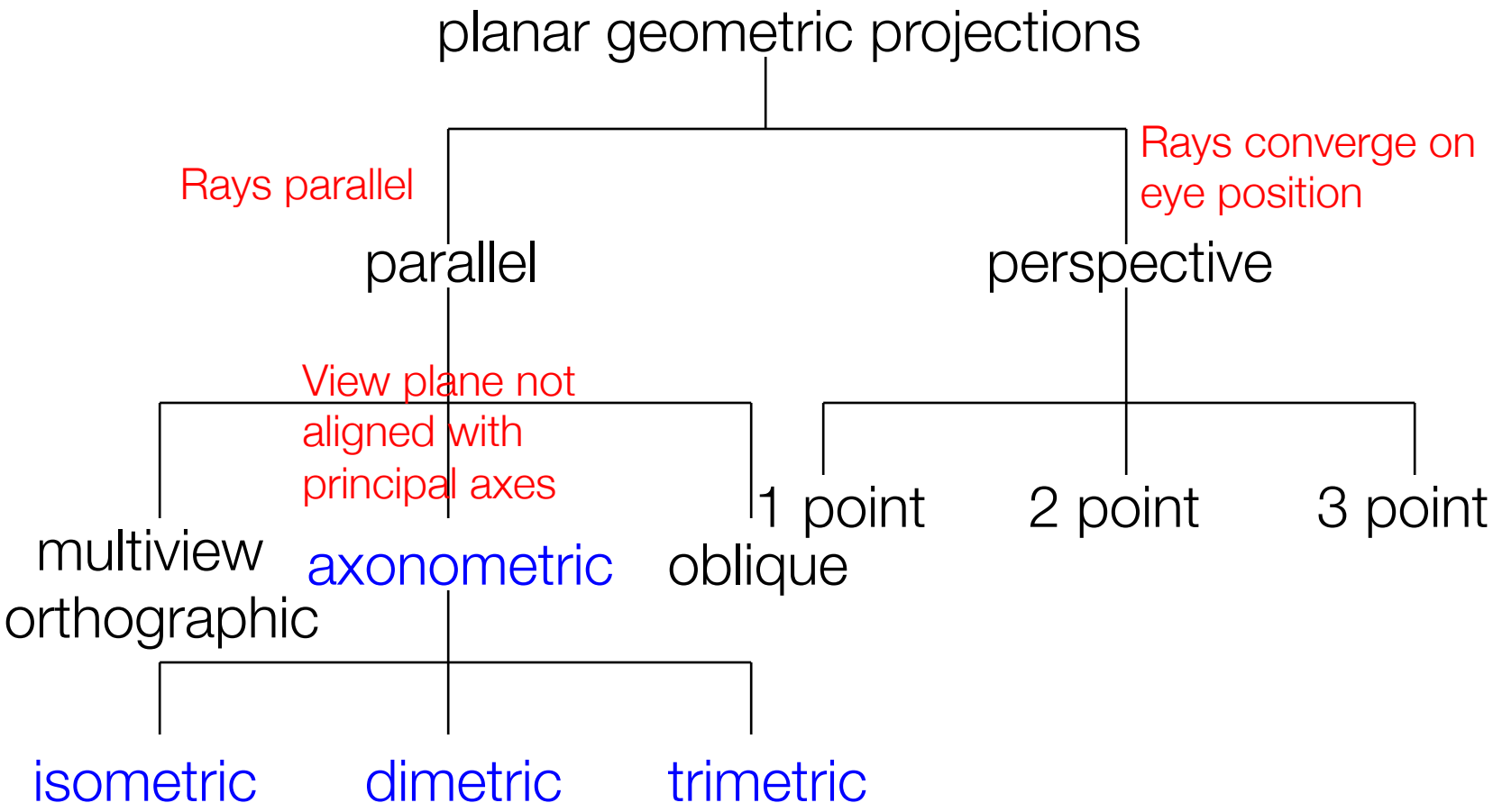
side



Advantages and Disadvantages

- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals
- Cannot see what object really looks like because many surfaces hidden from view
  - Often we add the isometric

Taxonomy of Planar Geometric Projections

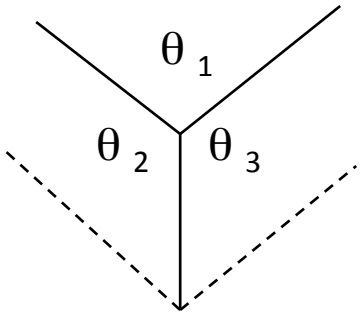


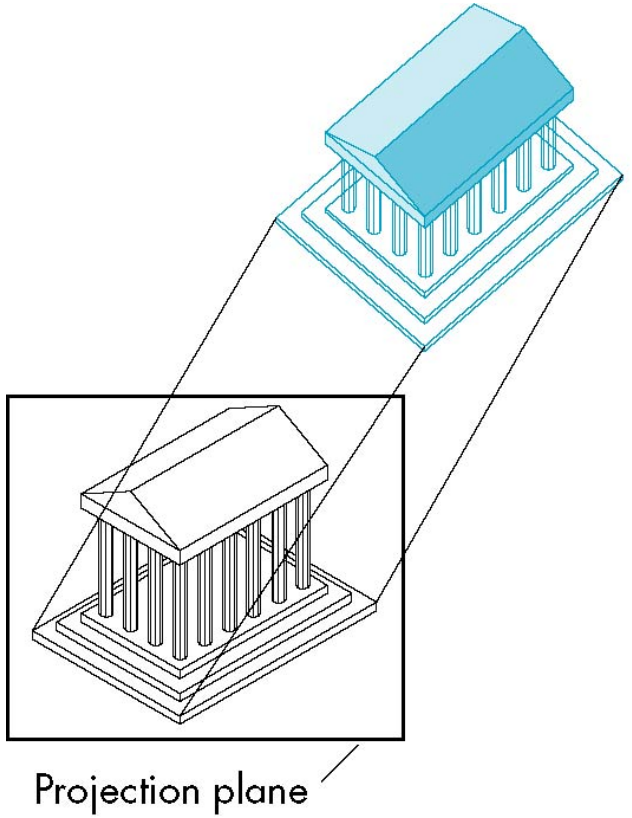
Axonometric Projections

- Axonometric means "to measure along axes"
- Allow projection plane to move relative to object (or the object is rotated along one or more of its axes relative to the plane of projection)

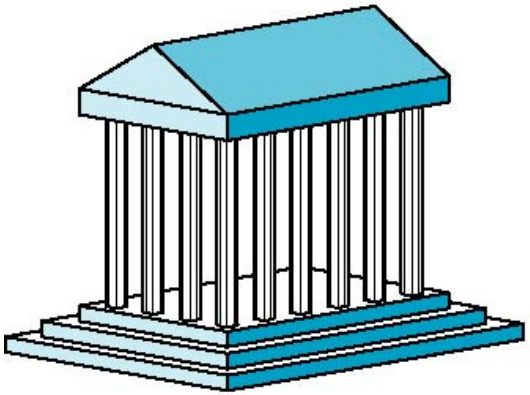
Classify by how many angles of a corner of a projected cube are the same

none: trimetric
two: dimetric
three: isometric

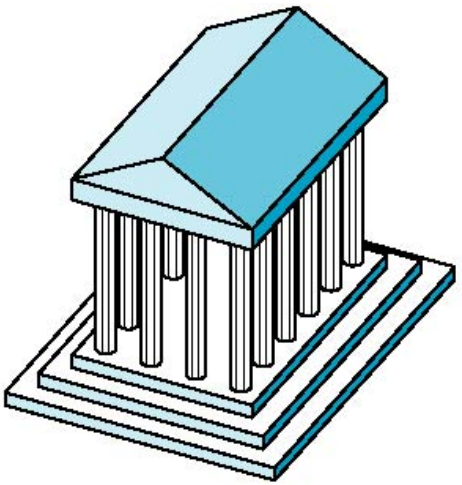




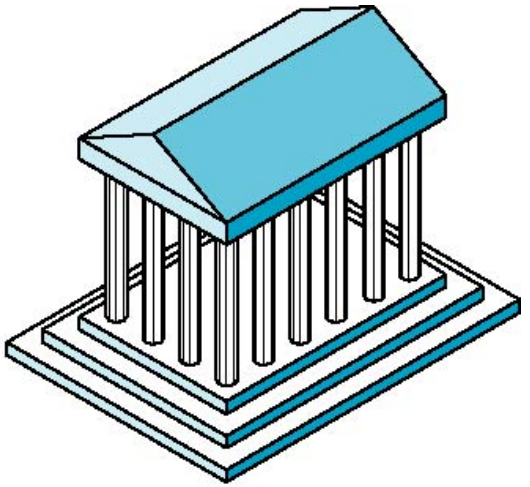
Types of Axonometric Projections



Dimetric

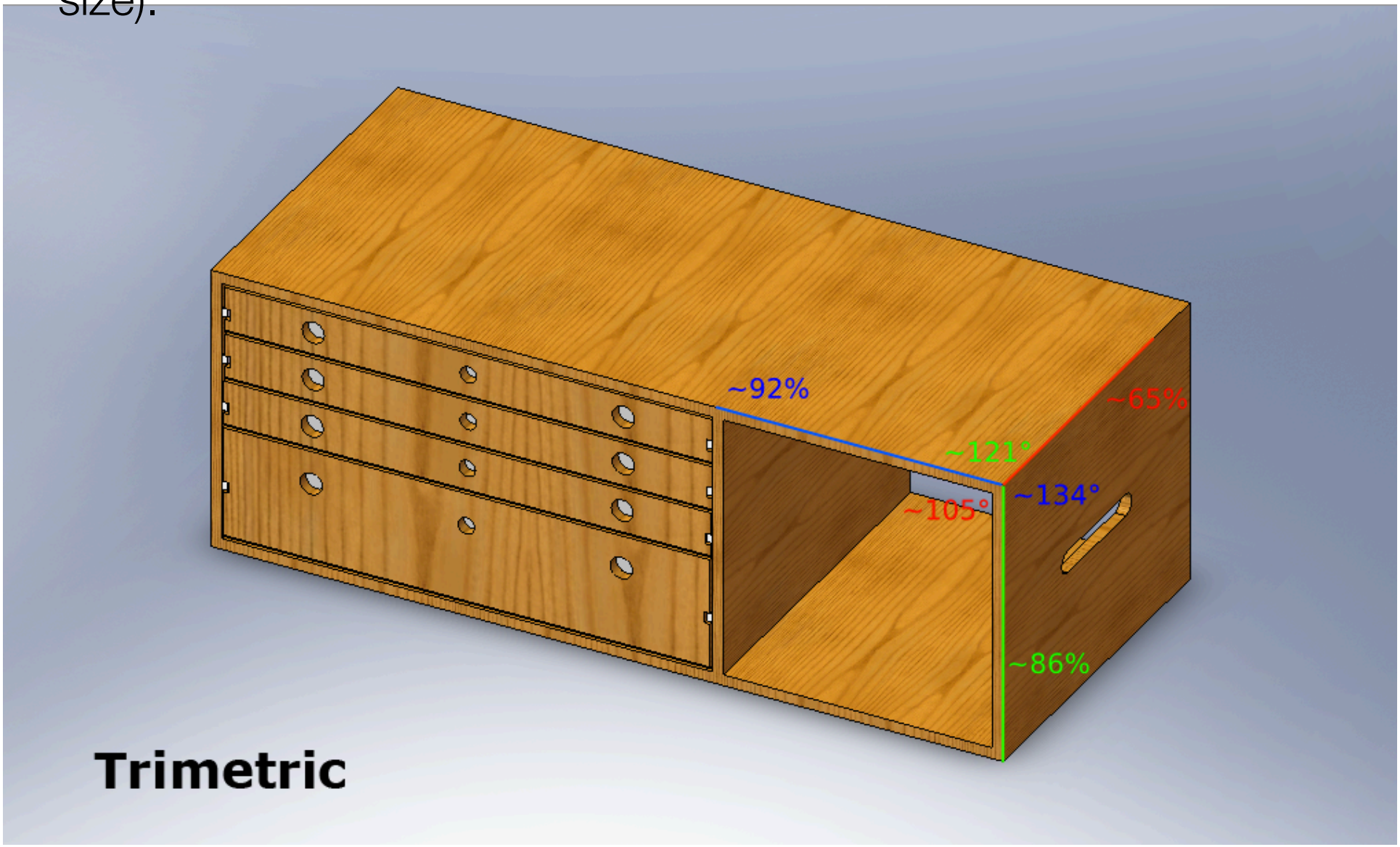


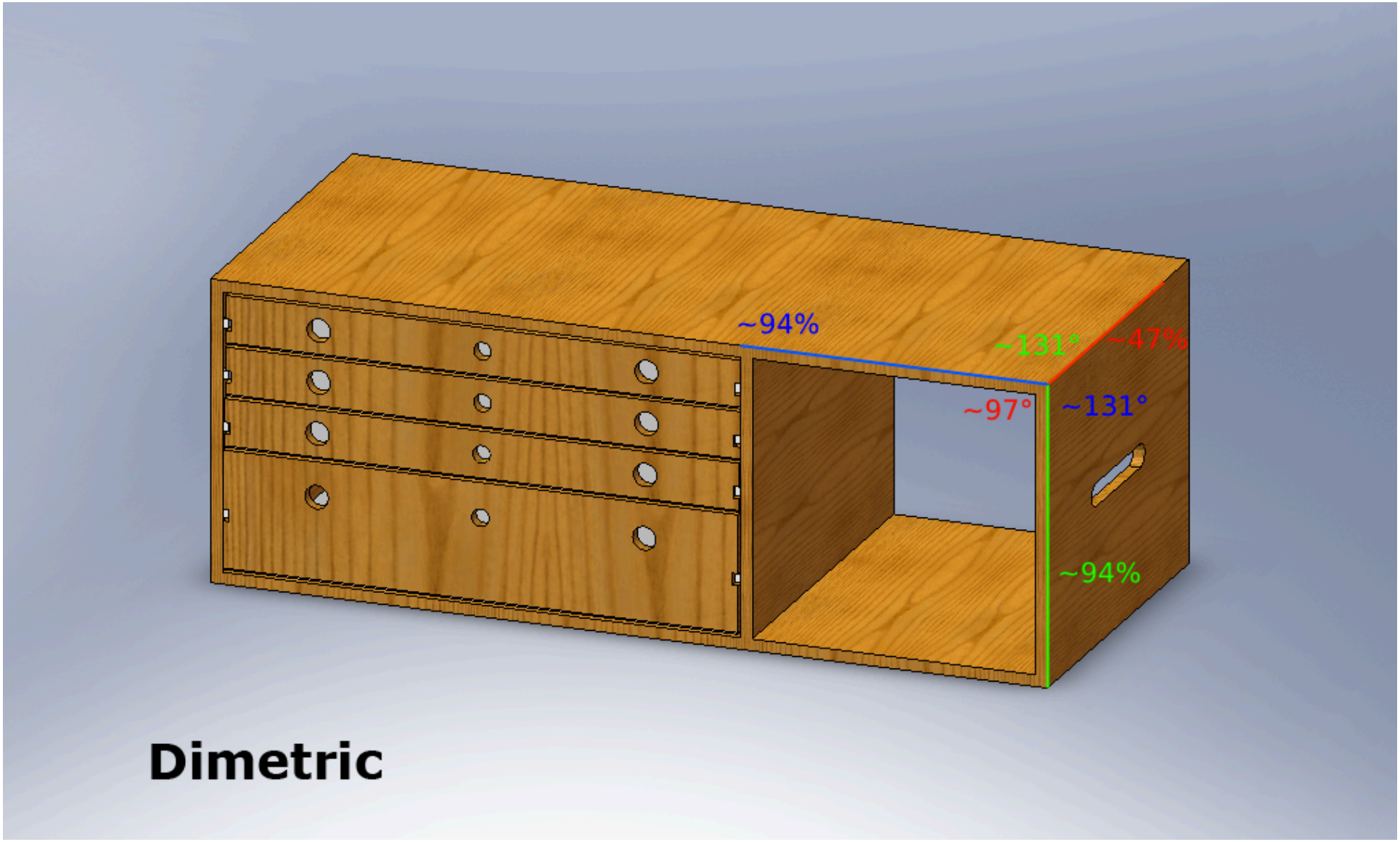
Trimetric



Isometric

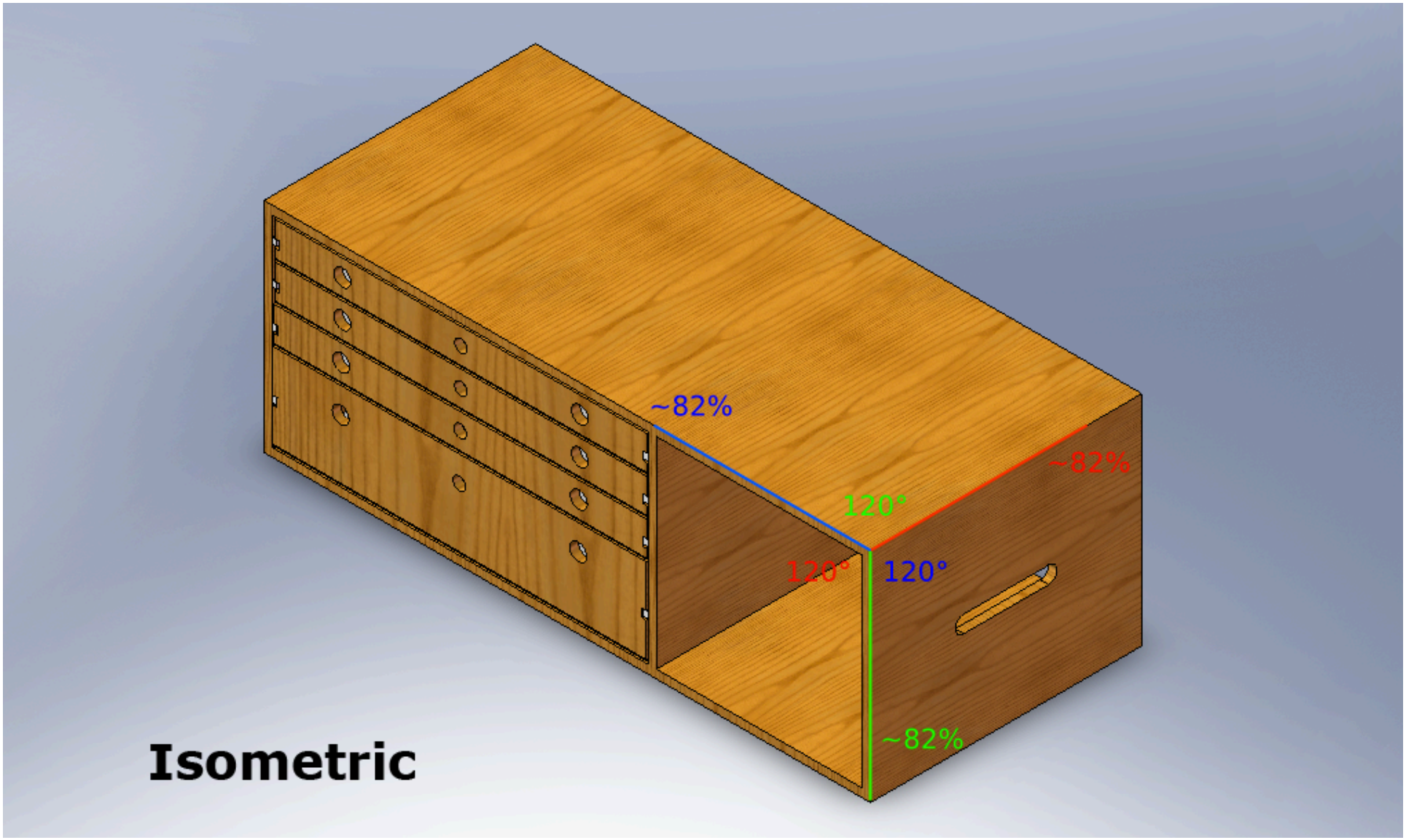
The degrees indicate the angles between the axes, while the percentages are the shortened lengths of the sides (e.g., 86% means that the dimension is displayed at 86 percent of its actual size).





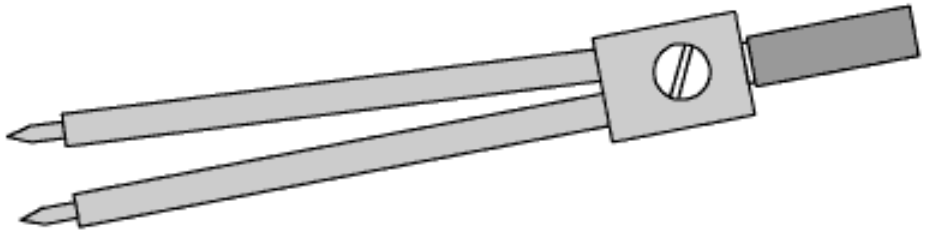
Dimetric

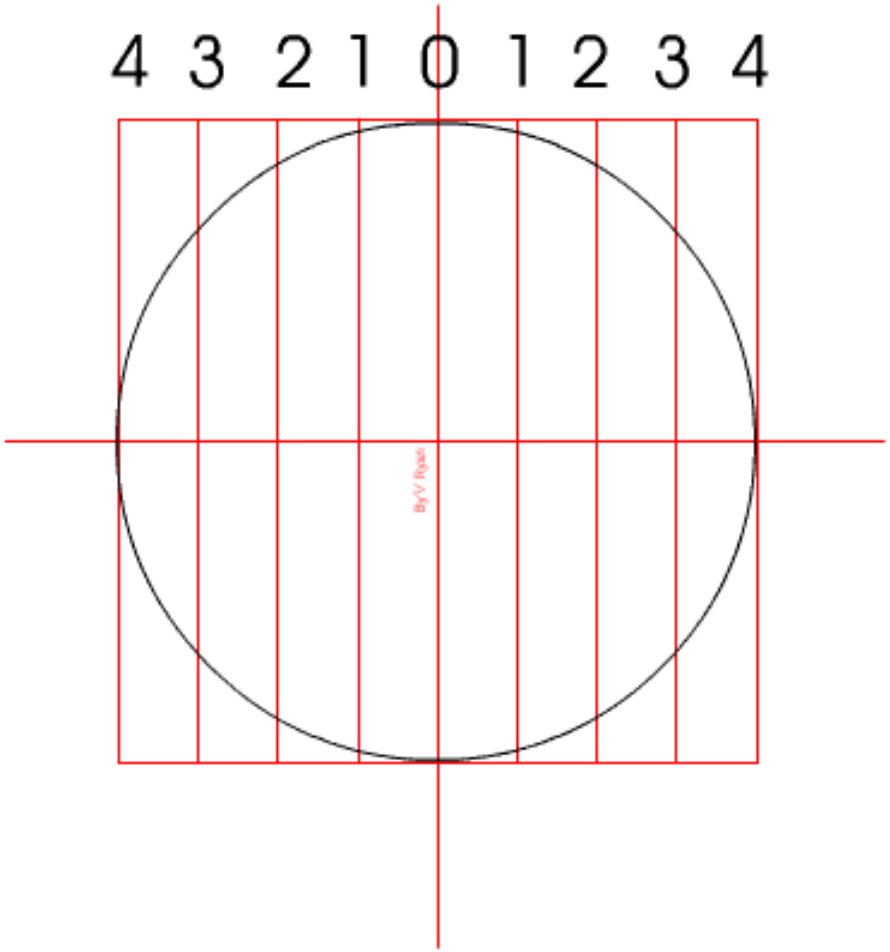
Isometric projection enables measurements to be read or taken directly from the drawing. It is mostly used in engineering drawing with the angle of 60°.

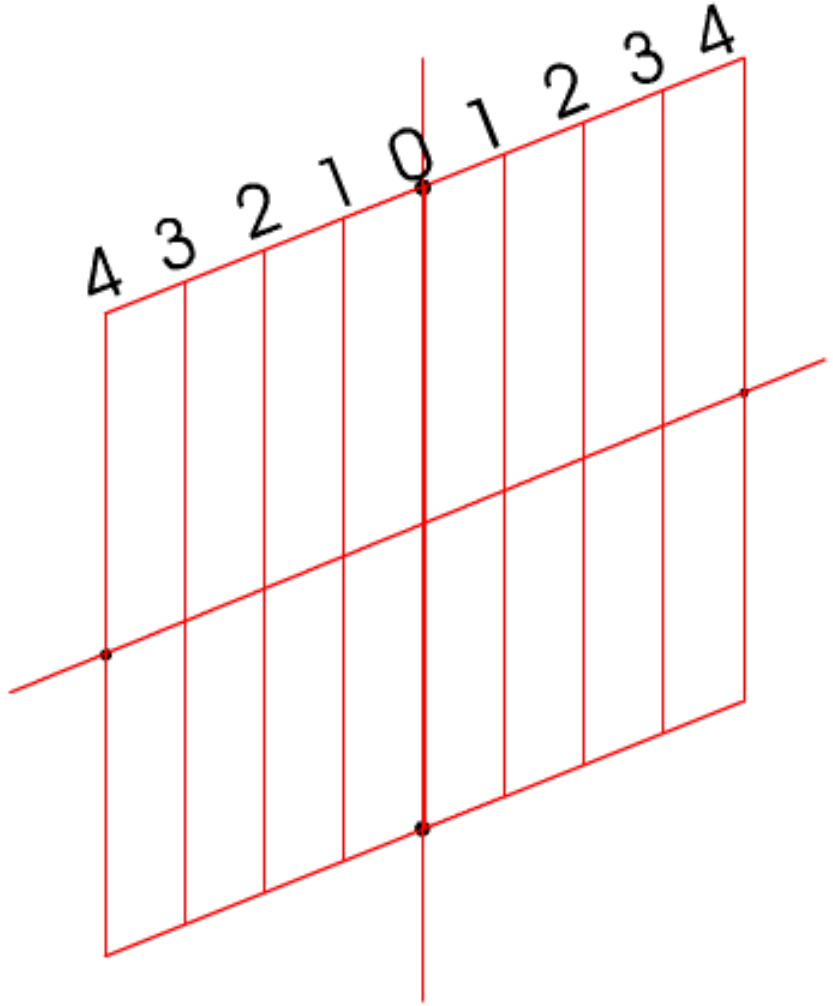


Advantages and Disadvantages

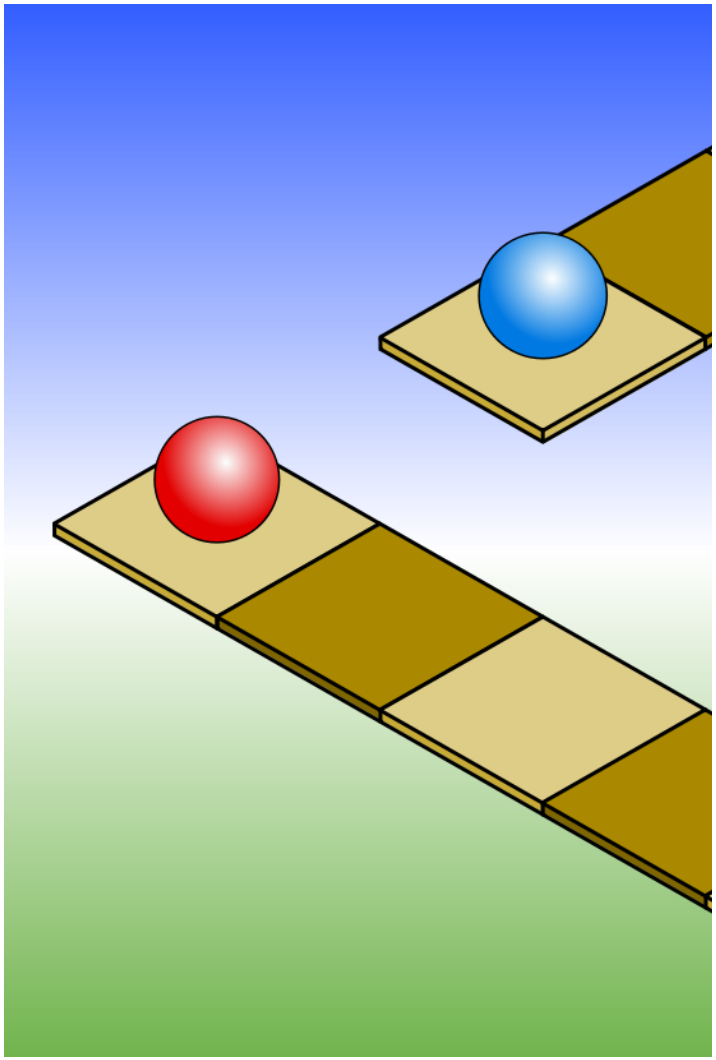
- Lines are scaled (foreshortened) but can find scaling factors
- Lines preserved but angles are not
  - Projection of a circle in a plane not parallel to the projection plane is an ellipse (see the animation next slide)
- Can see three principal faces of a box-like object
- Some optical illusions possible
  - Parallel lines appear to diverge
- Does not look real because far objects are scaled the same as near objects
- Used in CAD applications



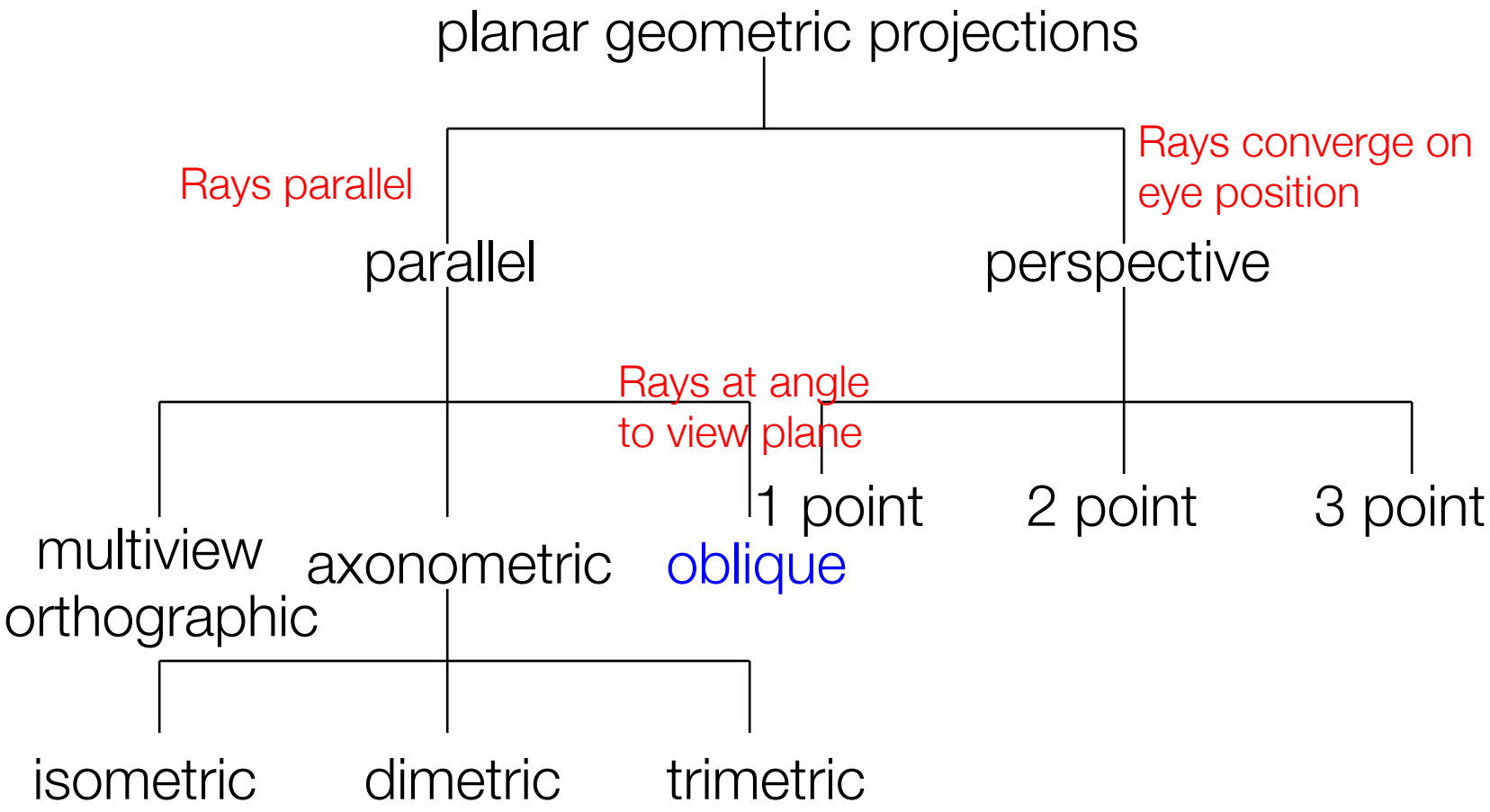

4 3 2 1 0 1 2 3 4


4 3 2 1 0 1 2 3 4

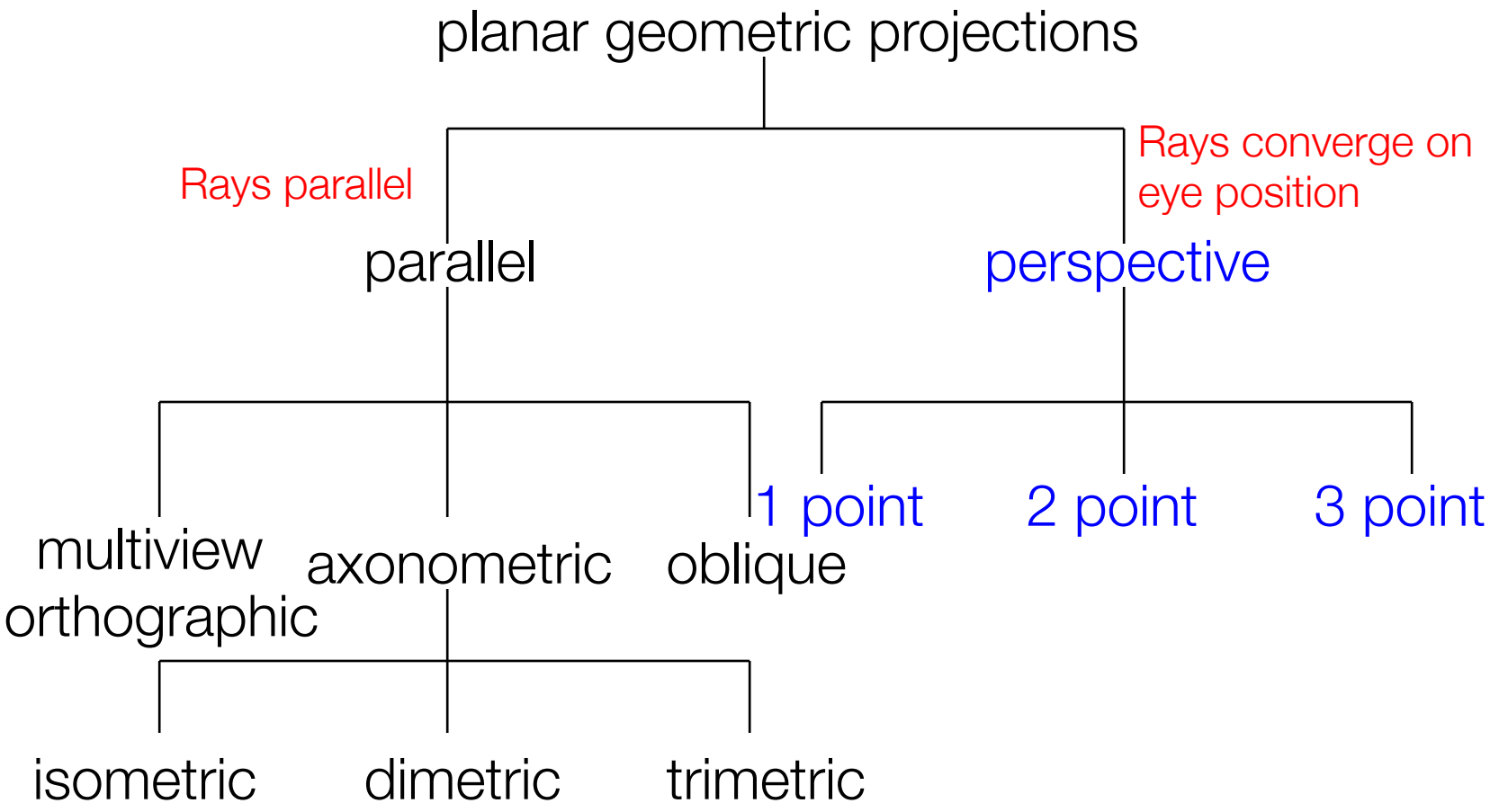
Visual Ambiguity of Axonometric Projection



Taxonomy of Planar Geometric Projections

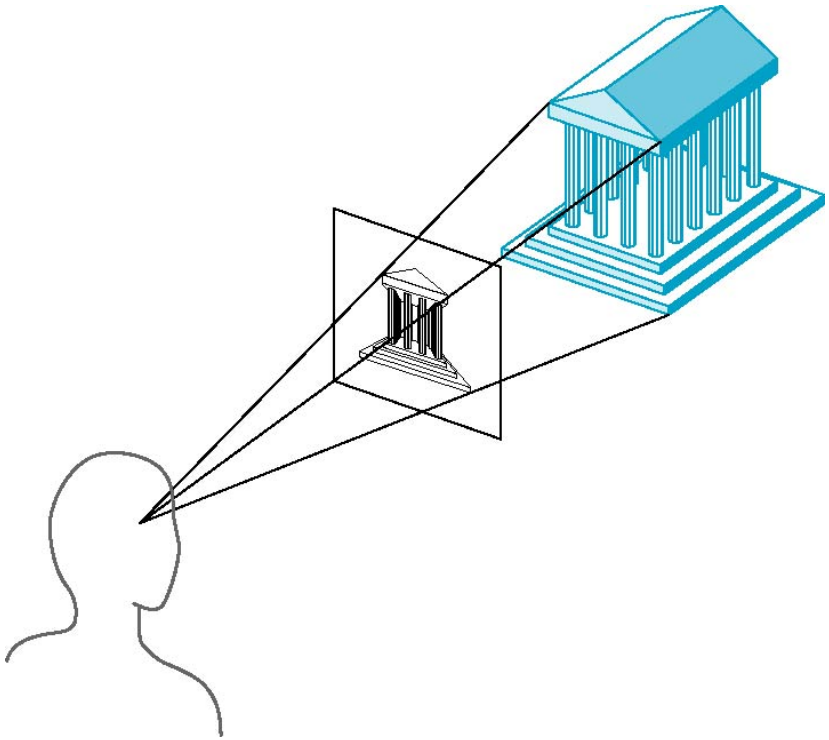


Taxonomy of Planar Geometric Projections



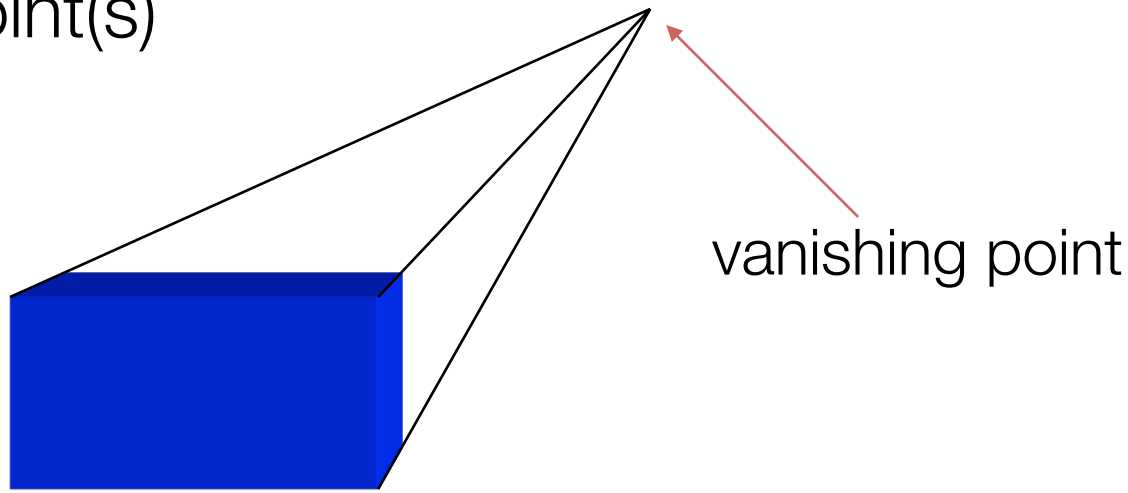
Perspective Projection

- Projectors converge at center of projection

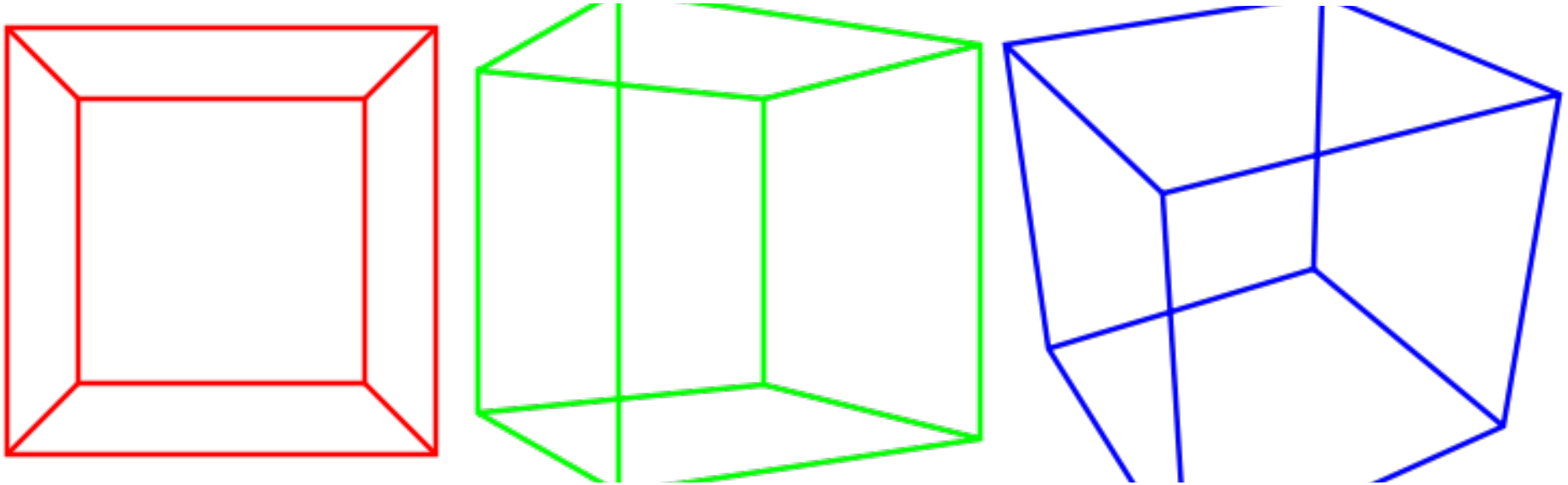


Vanishing Points

- Parallel lines (not parallel to the projection plan) on the object converge at a single point in the projection (i.e., vanishing point)
- Drawing simple perspectives by hand uses these vanishing point(s)



One-, Two- and Three-Point Perspective



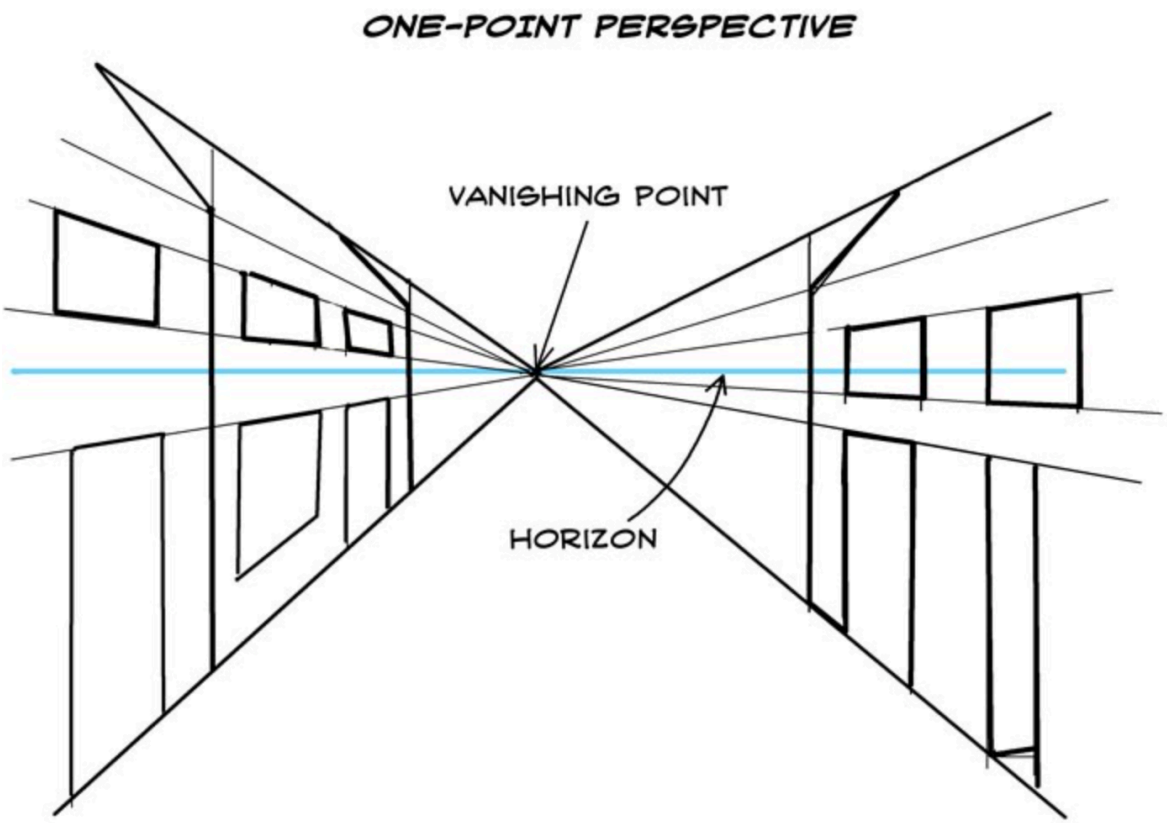
One-point

Two-point

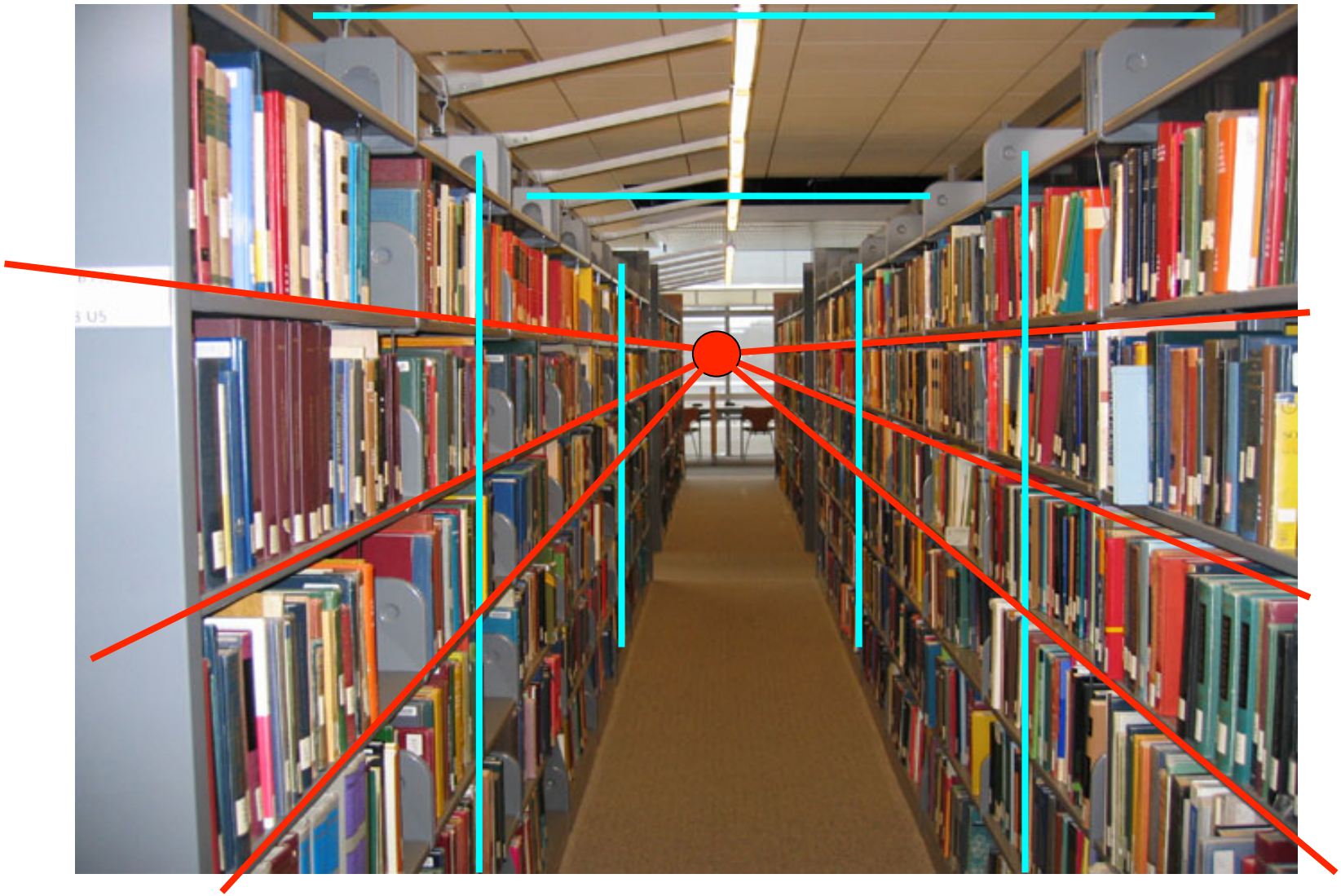
Three-point

One-Point Perspective

- One principal face parallel to projection plane
- This type of perspective is typically used for images of roads, railway tracks, hallways, or buildings so that the front is directly facing the viewer



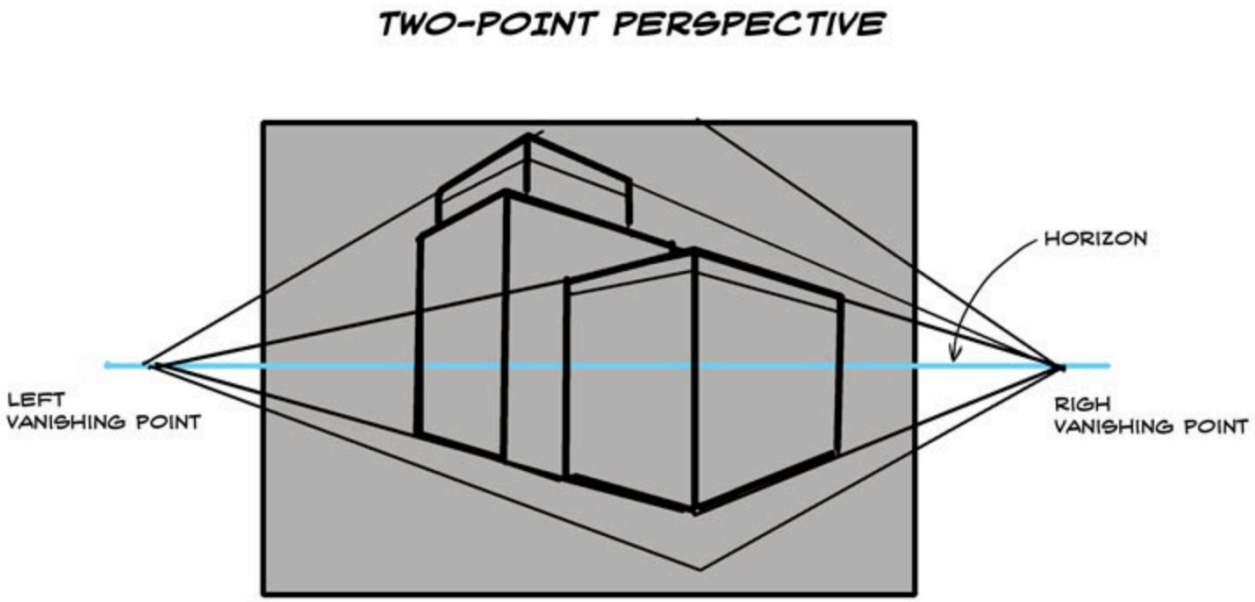
One-Point Perspective



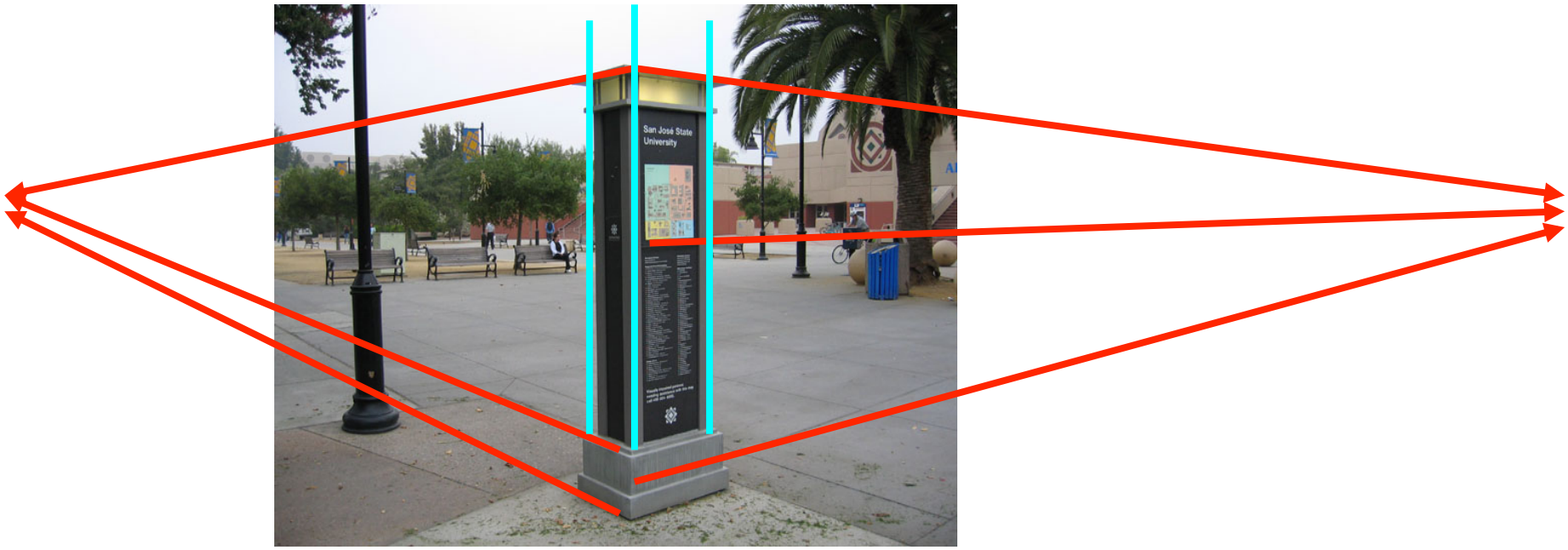
One-Point Perspective

Two-Point Perspective

- On principal direction parallel to projection plane
- Two-point perspective can be used to draw the same objects as one-point perspective, rotated: e.g., looking at the corner of a house

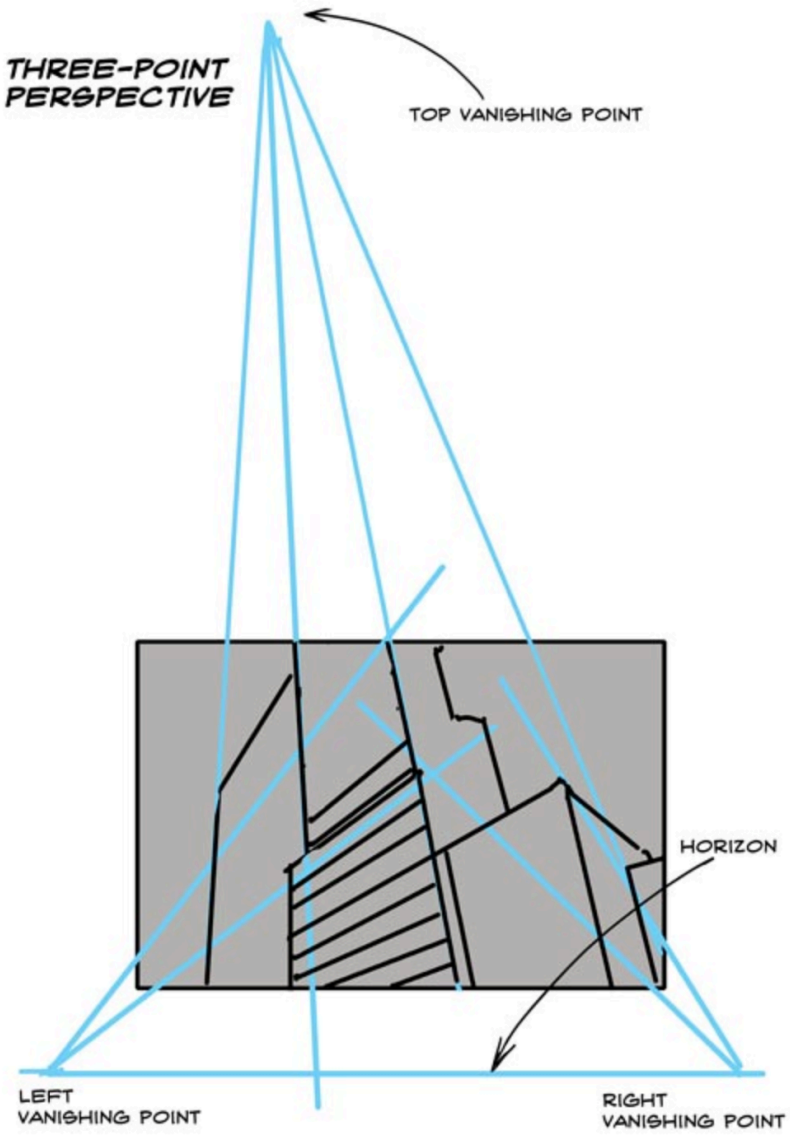


Two-Point Perspective

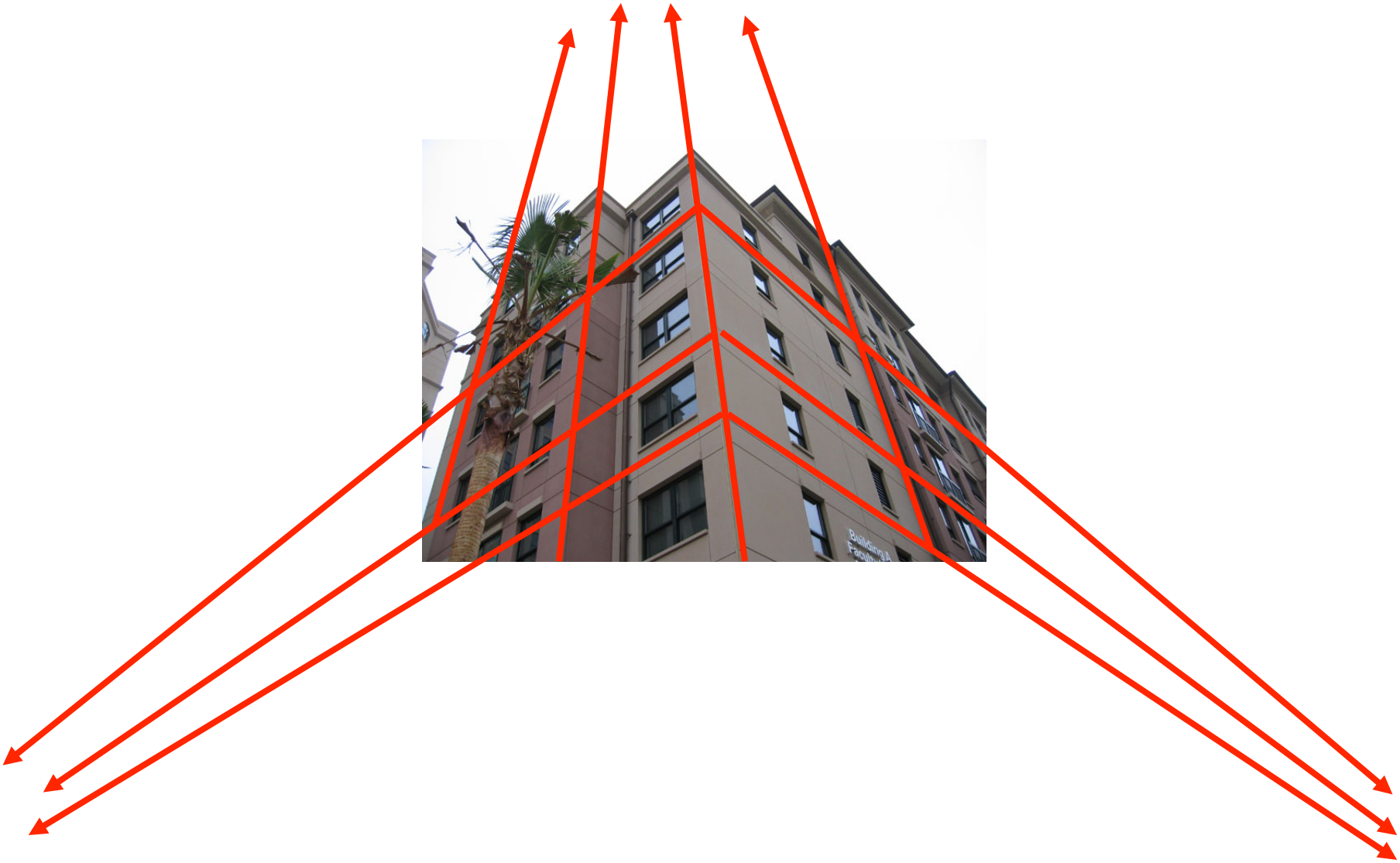


Three-Point Perspective

- No principal face parallel to projection plane
- Three-point perspective is usually used for buildings seen from above (or below)



Three-Point Perspective



Advantages and Disadvantages

- Objects further from viewer are projected smaller than the same sized objects closer to the viewer (diminution)
  - Looks realistic
- Equal distances along a line are not projected into equal distances (nonuniform foreshortening)
- Angles preserved only in planes parallel to the projection plane
- More difficult to construct by hand than parallel projections (but not more difficult by computer)

Computer Viewing

Objectives

- Introduce the mathematics of projection
- Introduce WebGL viewing functions in `MV.js`
- Look at alternate viewing APIs

From the Beginning

- In the beginning (traditional OpenGL)
  - Fixed function pipeline
  - Model-view and projection transformation
  - Predefined frames: model, object, camera, clip, NDC (normalized device coordinates), window
- After deprecation (programmable pipeline)
  - Pipeline with programmable shaders
  - No transformations
  - Clip, NDC window frames
- `MV.js` reintroduces original capabilities

Computer Viewing

- There are three aspects of the viewing process, all of which are implemented in the pipeline
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

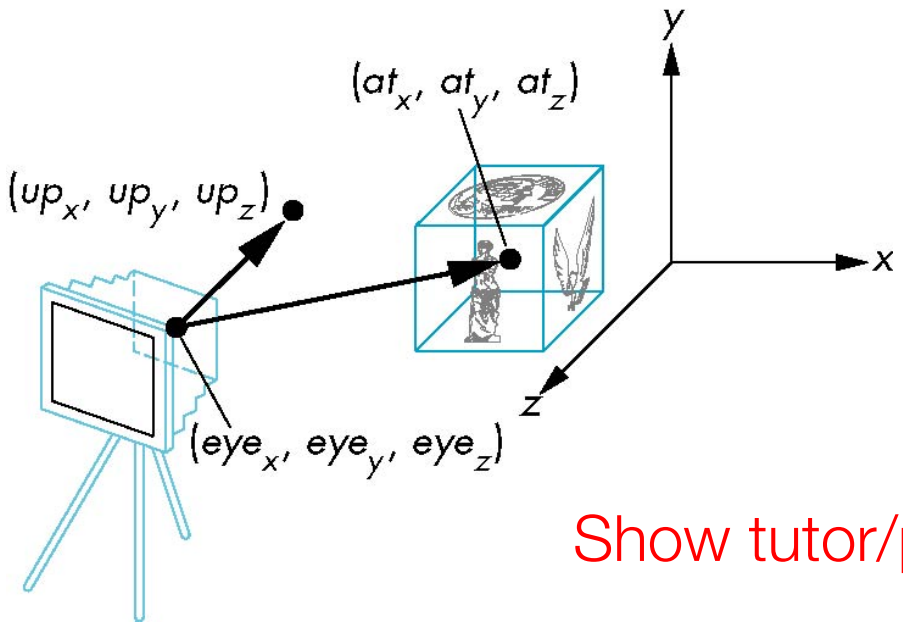
Viewing Transformation

- Transform the object from world to eye space
  - Construct an eye space coordinate frame
  - Construct a matrix to perform the coordinate transformation

The LookAt Function

- `MV.js` contains the function `lookAt` to form the required modelview matrix through a simple interface

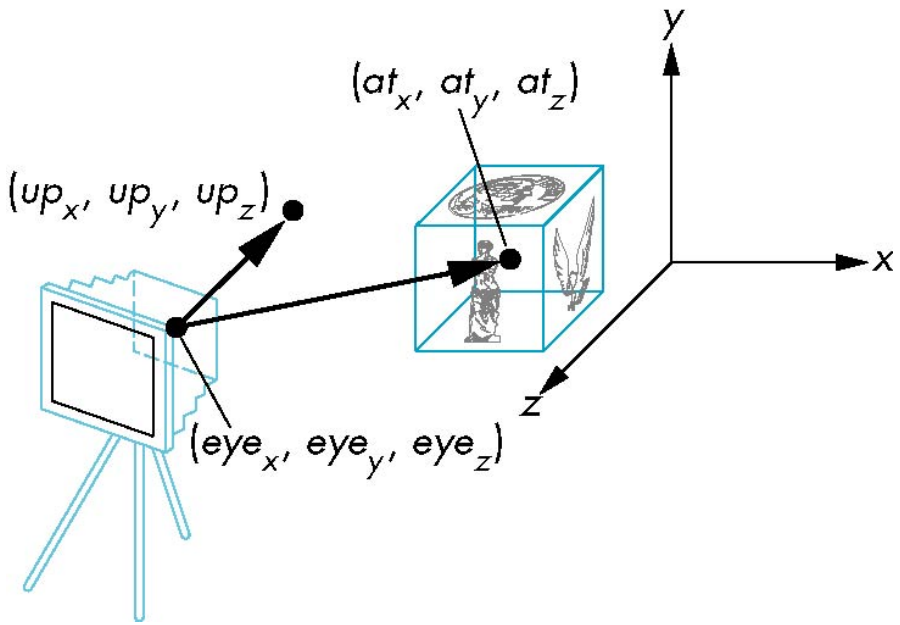
```
function lookAt(eye, at, up);
```



Show tutor/projection demo

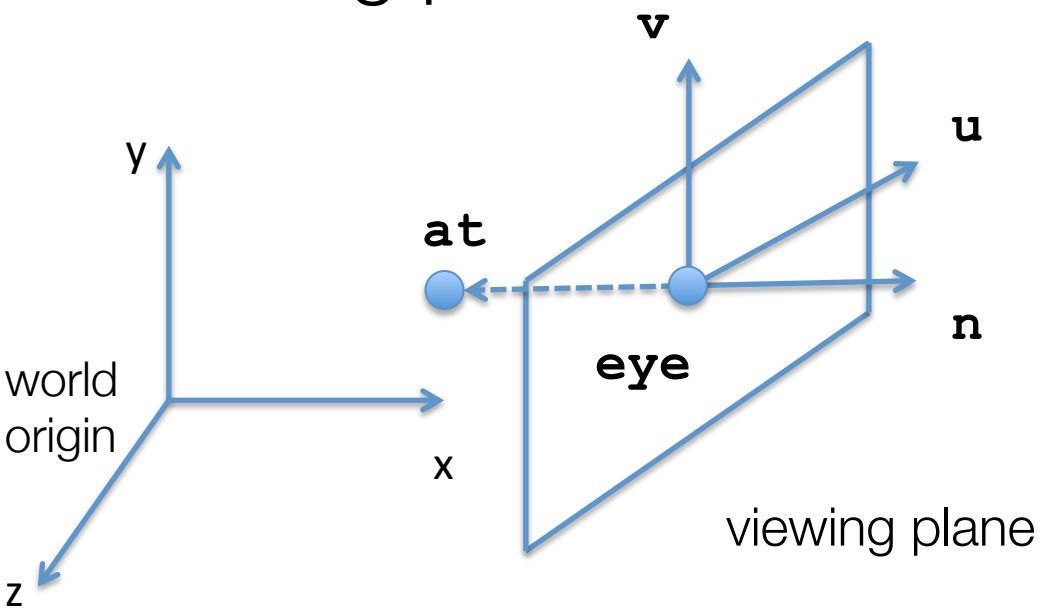
Eye Coordinate Frame (1)

- Known: eye position, center of interest (look-at point), view-up vector
- To find out: new origin and three basis vectors
- Assumption: the direction of view is orthogonal to the view plane where objects will be projected



Eye Coordinate Frame (2)

- Origin: eye position (easy!)
- Three basis vectors: one is the normal vector (**n**) of the viewing plane, the other two are the ones (**u** and **v**) that span the viewing plane



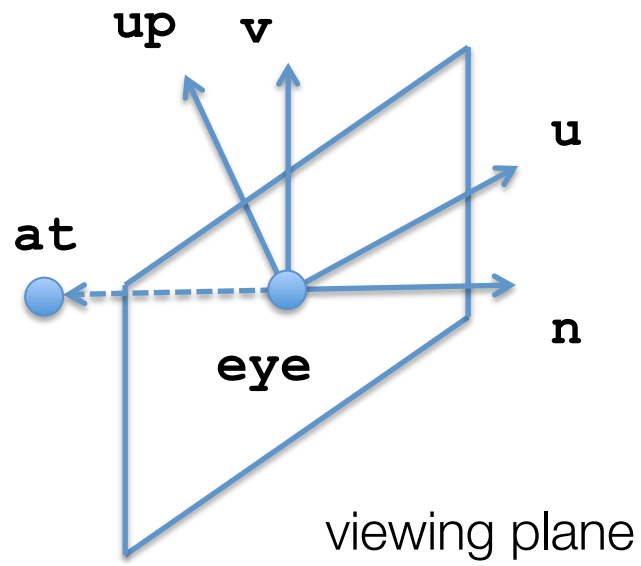
n is pointing away from the world

$$\mathbf{N} = \mathbf{eye} - \mathbf{at}$$

$$\mathbf{n} = \mathbf{N} / |\mathbf{N}|$$

Eye Coordinate Frame (3)

- How about **u** and **v**?



We can get **u** first:

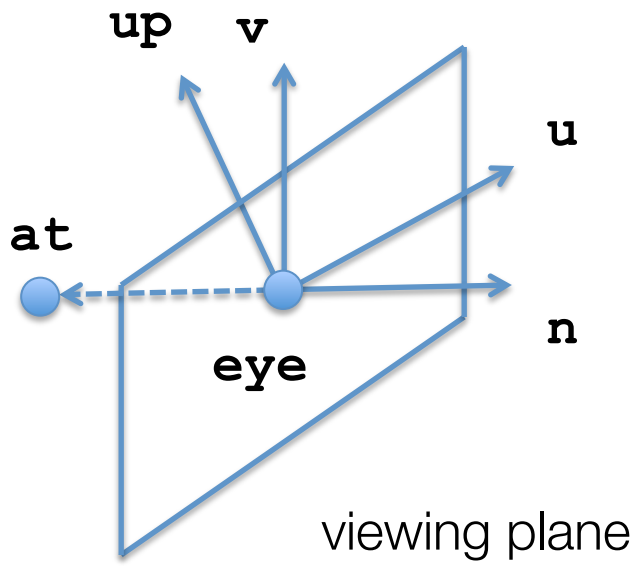
u is a vector that is perpendicular to the plane spanned by **n** and view up vector **up**

$$\mathbf{U} = \mathbf{up} \times \mathbf{n}$$

$$\mathbf{u} = \mathbf{U} / |\mathbf{U}|$$

Eye Coordinate Frame (4)

- How about **v**?



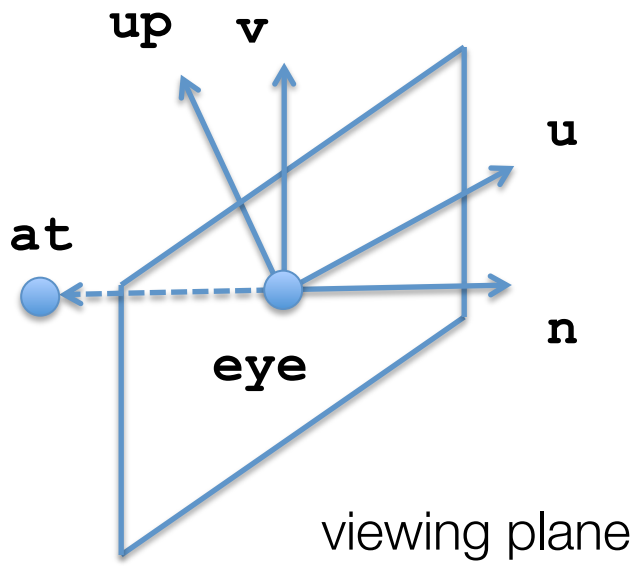
Knowing **n** and **u**, getting **u** is easy:

$$\mathbf{v} = \mathbf{n} \times \mathbf{u}$$

v is already normalized

Eye Coordinate Frame (5)

- Put it all together



Eye space origin:
(`eye.x`, `eye.y`, `eye.z`)

Basis vectors:

n = (**eye-at**) / |**eye-at**|
u = (**up**×**n**) / |**up**×**n**|
v = **n**×**u**