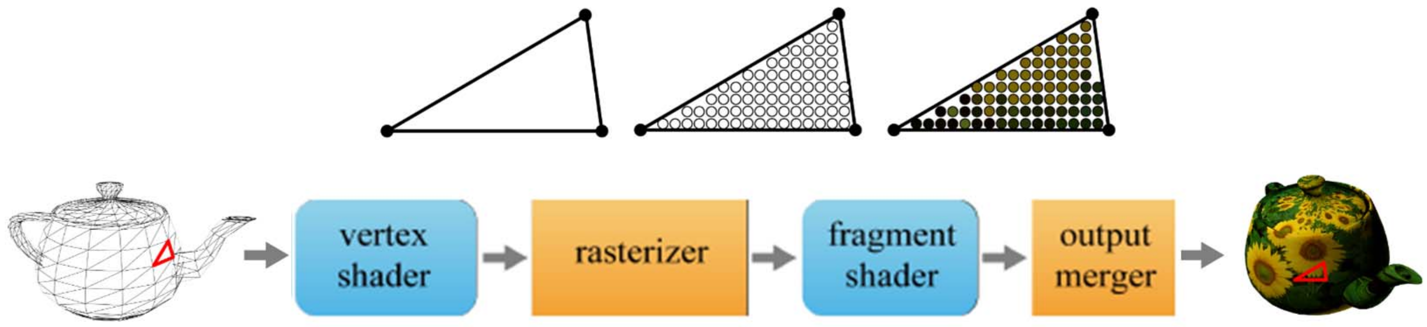
Chapter VII
Rasterizer

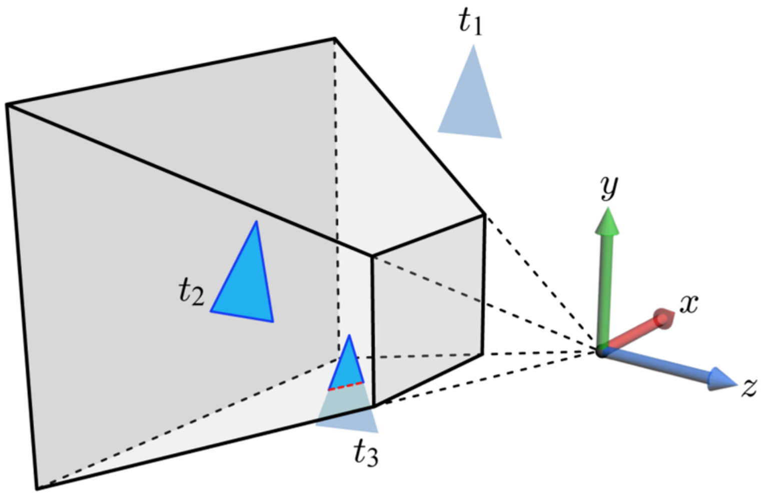
Rasterizer

- The vertex shader passes the clip-space vertices to the rasterizer, which performs the following:
  - Clipping
  - Perspective division
  - Back-face culling
  - Viewport transform
  - Scan conversion (rasterization in a narrow sense)



Clipping

- Clipping is performed in the clip space, but the following figure presents its concept in the camera space, for the sake of intuitive understanding.
  - ‘Completely outside’ triangles are discarded.
  - ‘Completely inside’ triangles are accepted.
  - ‘Intersecting’ triangles are clipped.



- As a result of clipping, vertices may be added to and deleted from the triangle.
- Clipping in the (homogeneous) clip space is a little complex but well-developed algorithm.

Perspective Division

- Unlike affine transforms, the last row of M_{proj} is not (0 0 0 1) but (0 0 -1 0). When M_{proj} is applied to (x,y,z,1), the w-coordinate of the transformed vertex is $-z$.

$$M_{proj} = \begin{pmatrix} \frac{cot(\frac{fovy}{2})}{aspect} & 0 & 0 & 0 \\ 0 & cot(\frac{fovy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} m_{11} & 0 & 0 & 0 \\ 0 & m_{22} & 0 & 0 \\ 0 & 0 & m_{33} & m_{34} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11}x \\ m_{22}y \\ m_{33}z + m_{34} \\ -z \end{pmatrix} \rightarrow \begin{pmatrix} -\frac{m_{11}x}{z} \\ -\frac{m_{22}y}{z} \\ -m_{33} - \frac{m_{34}}{z} \\ 1 \end{pmatrix}$$

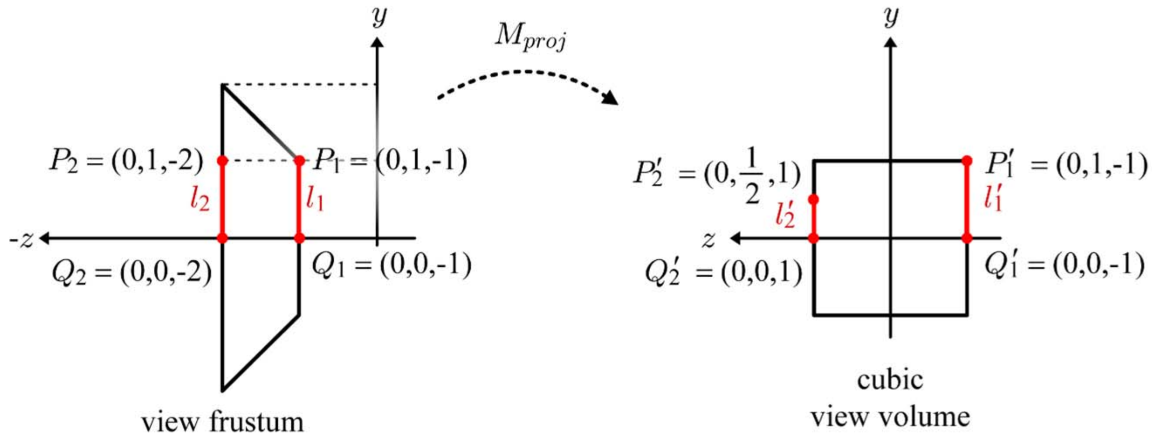
- In order to convert from the homogeneous (clip) space to the Cartesian space, each vertex should be divided by its w-coordinate (which equals $-z$).

Perspective Division (cont'd)

- Note that $-z$ is a positive value representing the *distance* from the *xy*-plane of the camera space. Division by $-z$ makes distant objects smaller. It is *perspective division*. The result is said to be in NDC (normalized device coordinates).

$$M_{proj} = \begin{pmatrix} \frac{cot(\frac{fovy}{2})}{aspect} & 0 & 0 & 0 \\ 0 & cot(\frac{fovy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\boxed{\begin{array}{l} fovy = \frac{\pi}{2} \\ aspect = 1 \\ n = 1 \\ f = 2 \end{array}} \Rightarrow M_{proj} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -3 & -4 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

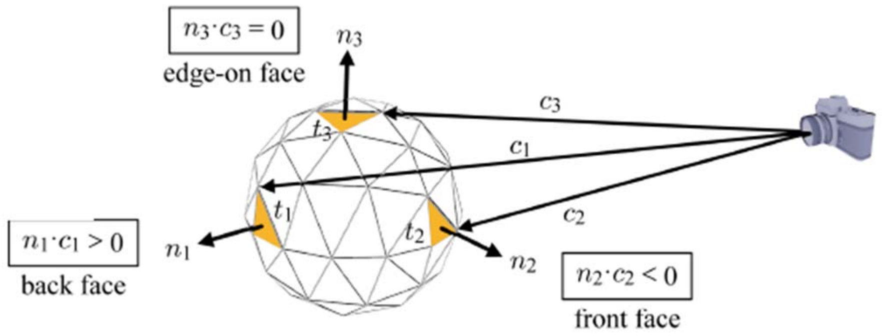


$$M_{proj}P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -3 & -4 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ -1 \\ 1 \end{pmatrix} = P_1' \qquad M_{proj}P_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -3 & -4 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 2 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ \frac{1}{2} \\ 1 \\ 1 \end{pmatrix} = P_2'$$

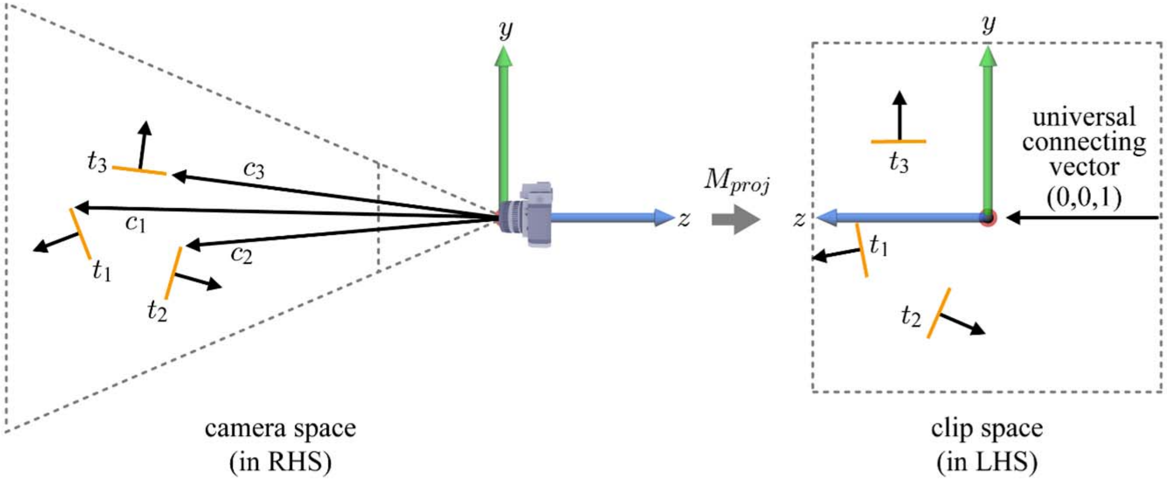
$$M_{proj}Q_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -3 & -4 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 1 \end{pmatrix} = Q_1' \qquad M_{proj}Q_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -3 & -4 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 2 \\ 2 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = Q_2'$$

Back-face Culling

- The polygons facing away from the viewpoint of the camera are discarded. Such polygons are called *back-faces*. (The polygons facing the camera are called *front-faces*.)

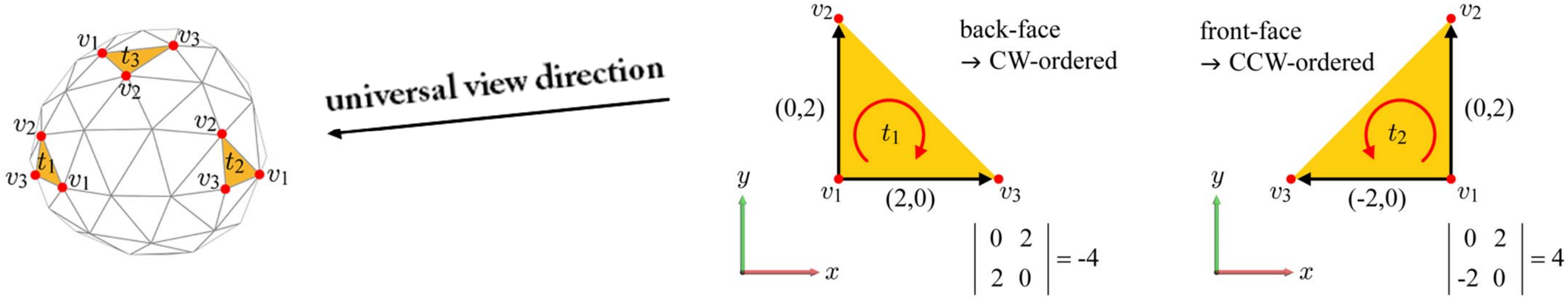


- The projection transform defines a universal connecting vector parallel to the *z*-axis, which is identical to the view direction.



Back-face Culling (cont'd)

- Viewing a triangle along the universal view direction is equivalent to orthographically projecting the triangle onto the *xy*-plane.
- A 2D triangle with CW-ordered vertices is a back-face, and a 2D triangle with CCW-ordered vertices is a front-face.



- Compute the following determinant, where the first row represents the 2D vector connecting v_1 and v_2, and the second row represents the 2D vector connecting v_1 and v_3.

$$\begin{vmatrix} (x_2 - x_1) & (y_2 - y_1) \\ (x_3 - x_1) & (y_3 - y_1) \end{vmatrix}$$

  - If negative, CW and so back-face.
  - If it is positive, CCW and so front-face.
  - If 0, edge-on face.

Back-face Culling (cont'd)

- The back-faces are not always culled.
  - Consider rendering a translucent sphere. For the back-faces to show through the front-faces, no face will be culled.
  - Another example is culling only the front faces of a hollow sphere. Then the cross-section view of the sphere will be obtained.

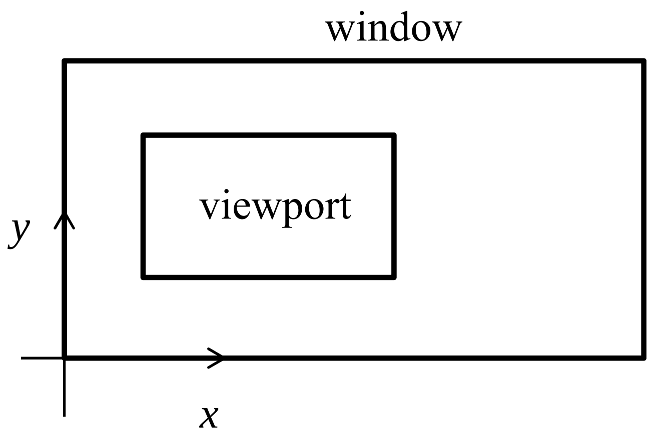
- Various GL capabilities are enabled by `glEnable` and disabled by `glDisable`.
- `void glEnable(GLenum cap)` & `void glDisable(GLenum cap)`
- An example is `glEnable(GL_CULL_FACE)`, which enables face culling.
- The default value is `GL_FALSE.`

Back-face Culling (cont'd)

- When face culling is enabled, `glCullFace()` specifies whether front or back faces are culled. It accepts the following symbolic constants:
  - `GL_FRONT`
  - `GL_BACK`
  - `GL_FRONT_AND_BACK`
- The default value is `GL_BACK`, and back faces are culled.
- Then, `glFrontFace()` specifies the vertex order of front faces. It accepts the following:
  - `GL_CW` vertax 순서가 바뀐경우 opengl에 front-face의 방향 시계방향으로 설정
  - `GL_CCW` vertax 순서가 안바뀐경우 opengl에 front-face의 방향 반시계방향으로 설정
- The default value is `GL_CCW`. 디폴트는 반시계 방향

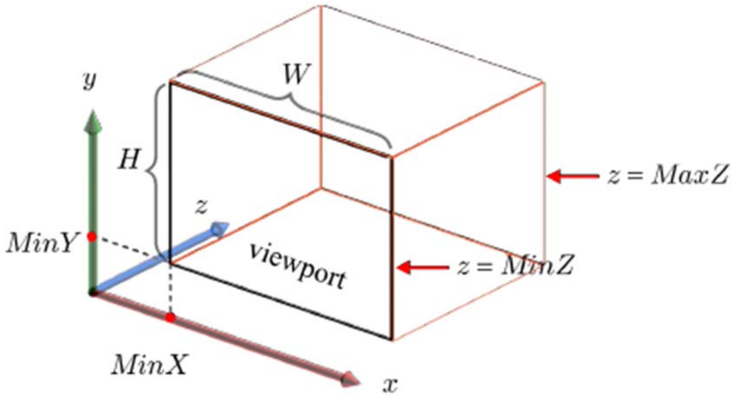
Viewport

- A window at the computer screen is associated with its own screen space.
- A viewport defines a screen-space rectangle into which the scene is projected. The rectangle is not necessarily the entire window, but can be a sub-area of the window.



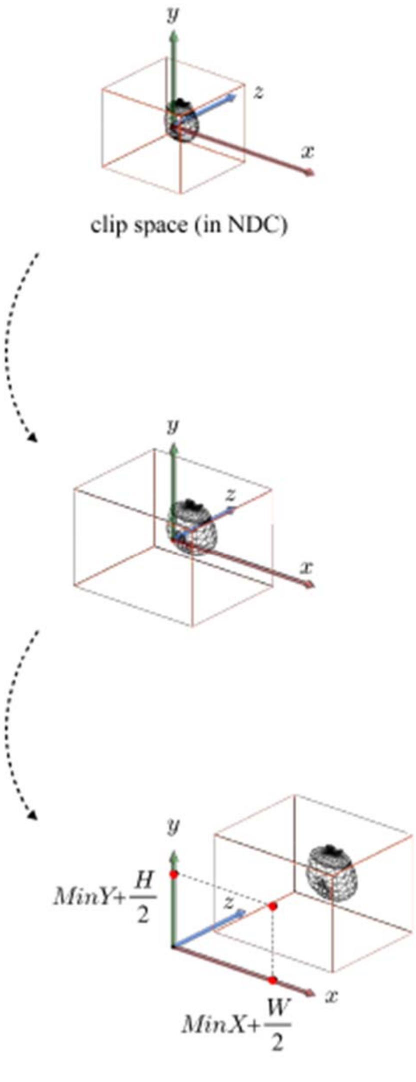
Viewport

- In reality, the screen space is 3D and so is the viewport, where the *z*-axis goes into the window.



- `void glViewport(GLint MinX, GLint MinY, GLsizei W, GLsizei H)`
  - (`MinX,MinY`) specify the screen coordinates of the viewport's lower left corner in pixels
  - `W` and `H` specify the width and height of viewport in pixels. These values must be > 0
- `void glDepthRangef(GLclampf MinZ, GLclampf MaxZ)`
  - (`MinZ,MaxZ`) specify the desired depth range. Default values are 0.0 and 1.0, respectively.

Viewport Transform



screen space

$$\begin{pmatrix} \frac{W}{2} & 0 & 0 & 0 \\ 0 & \frac{H}{2} & 0 & 0 \\ 0 & 0 & \frac{MaxZ-MinZ}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

scaling

$$\begin{pmatrix} 1 & 0 & 0 & MinX+\frac{W}{2} \\ 0 & 1 & 0 & MinY+\frac{H}{2} \\ 0 & 0 & 1 & \frac{MaxZ+MinZ}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

translation

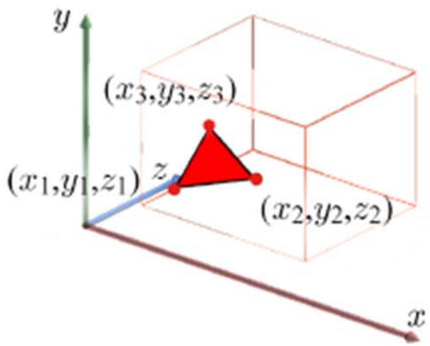
$$\begin{pmatrix} \frac{W}{2} & 0 & 0 & MinX+\frac{W}{2} \\ 0 & \frac{H}{2} & 0 & MinY+\frac{H}{2} \\ 0 & 0 & \frac{MaxZ-MinZ}{2} & \frac{MaxZ+MinZ}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

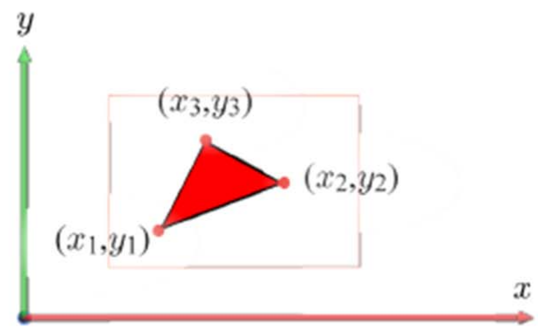
In most applications, *MinZ* and *MaxZ* are set to 0.0 and 1.0, respectively, and both of *MinX* and *MinY* are zero.

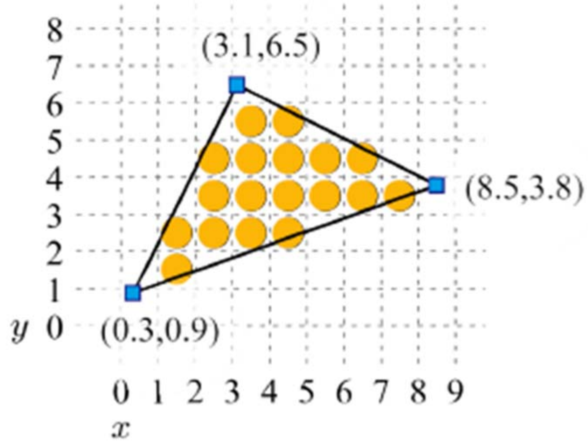
$$\begin{pmatrix} \frac{W}{2} & 0 & 0 & \frac{W}{2} \\ 0 & \frac{H}{2} & 0 & \frac{H}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scan Conversion

- Each screen-space triangle is rasterized into a set of *fragments* at the screen-space pixel locations covered by the triangle.
- The per-vertex attributes are *interpolated* to determine the per-fragment attributes at each pixel location.

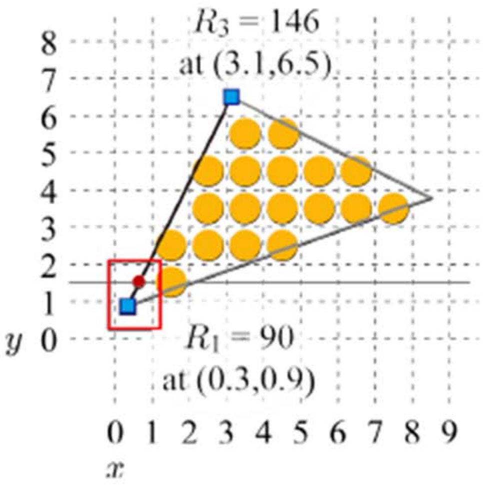






- The per-vertex attributes usually do not include RGB color but include normals and texture coordinates.
- Just for the convenience of presentation, however, let's assume color attributes and use *R* color for scan conversion.

Scan Conversion (cont'd)



$$\Delta y = 6.5 - 0.9 = 5.6$$

$$\Delta R = 146 - 90 = 56$$

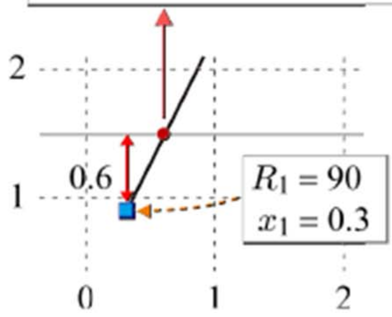
$$\frac{\Delta R}{\Delta y} = \frac{56}{5.6} = 10$$

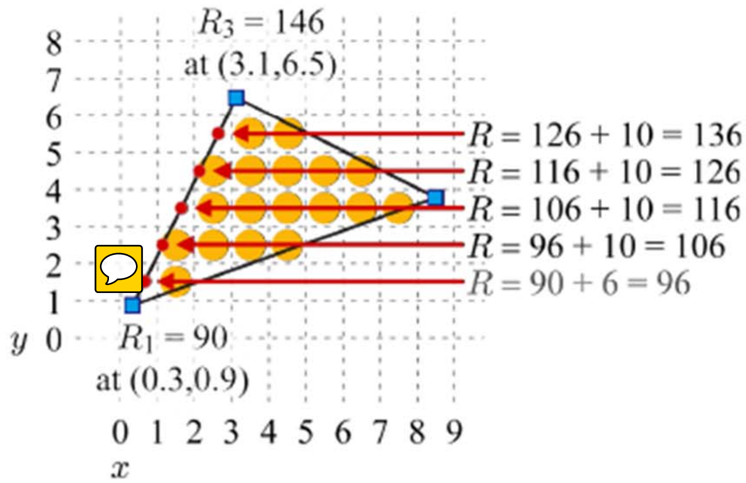
$$\Delta x = 3.1 - 0.3 = 2.8$$

$$\frac{\Delta x}{\Delta y} = \frac{2.8}{5.6} = 0.5$$

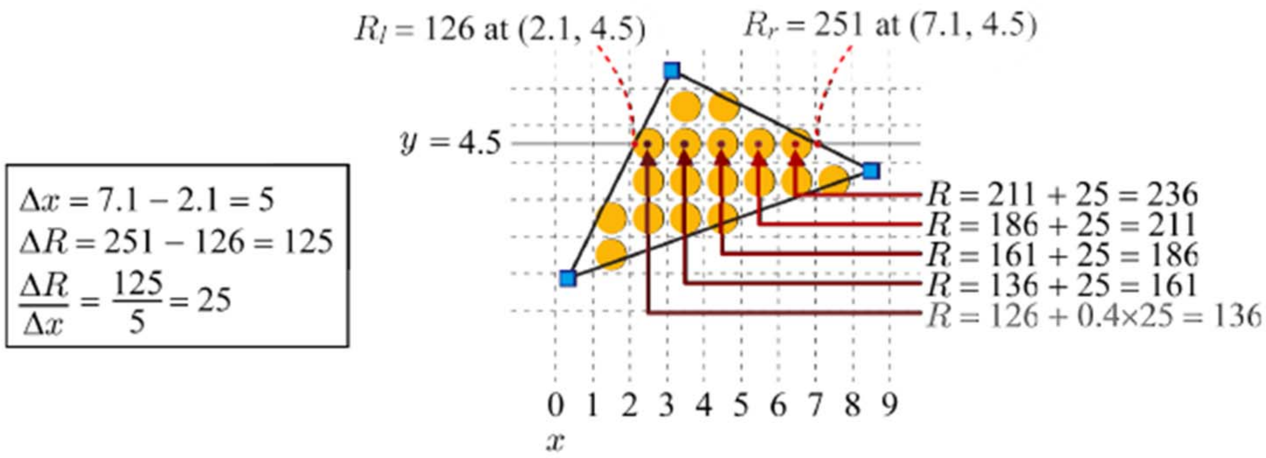
$$R = R_1 + 0.6\,\frac{\Delta R}{\Delta y} = 90 + 0.6 \times 10 = 96$$

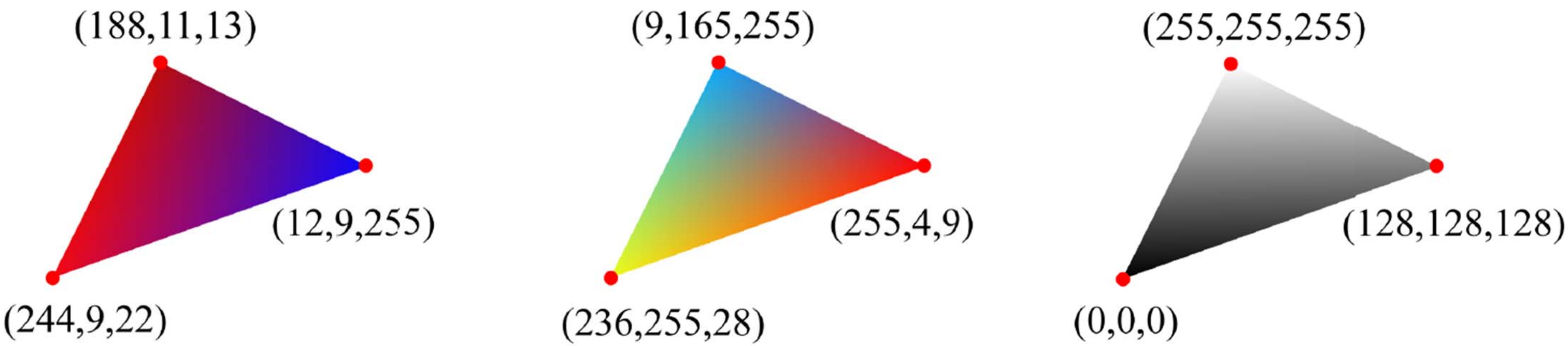
$$x = x_1 + 0.6\,\frac{\Delta x}{\Delta y} = 0.3 + 0.6 \times 0.5 = 0.6$$





Scan Conversion (cont'd)





Scan Conversion (cont'd)

- In general, what are actually interpolated are not colors but vertex normals and texture coordinates.
- Given (n_x, n_y, n_z) per vertex, each of n_x, n_y, and n_z is independently interpolated.
- Then we have the following interpolated normals.