CPT205 Final

Lecture 1 Hardware and Software

Graphic Handware

Graphics devices

- Input devices
  - Mouse
  - Joystick
  - Tablets
  - Virtual Reality Trackers
  - Light pens
  - Voice Systems ...
- Processing Devices
  - GPU / Graphics Cards
- Output Devices
  - Head-Mounted Displays(HMDs)
  - Head-Tracked Displays(HTDs)
  - Laptop LCD ...
- Framebuffer
  A block of memory, dedicated to graphics output, that holds the content of what will be displayed
  一块专用于图形输出的内存，保存将要显示的内容
  1 bit get 2 colors, just for 0 & 1 ｜8 bits get 2^8 colors
  - Pixel: an element of the framebuffer
  - bit depth: number of bits allocated per pixel in a buffer
    - Remember, we are asking "how much memory we allocate to store the color at each pixel" --common answer: 32bits RGBA
      - 32bits per pixel(true color)
        - 8 bits for red, green blue, and alpha
        - total colors: 16,777,216
  - Framebuffer $\rightarrow$ monitor
    - the values in the framebuffer are converted from a digital(1s and 0s representation, the bits)to an analog signal that goes to the monitor
      帧缓冲器中的值是从数字表示转换为模拟信号，输出到显示器

        - this is done automatically (not controlled by your code), and the conversion can be done while writing to the frame buffer

- Pixels
  The most basic addressable image element in a screen
    - CRT - Color triad (RGB phosphor dots)
    - LCD -Single color element
- Screen Resolution
  Measure of number of pixels on a screen(m by n)
    - m Horizontal screen resolution
    - n Vertical screen resolution
- Video formats
    - NTSC | PAL | VGA | SVGA | RGB | Interlaced
- Raster display 光栅显示器
    - Cathode Ray Tubes(CRTs) 阴极射线管: 曾经非常普遍但体积庞大
        - Strong electrical fields and high voltage
        - Very good resolution
        - Heavy, not flat
    - Liquid Crystal Displays(LCDs) 液晶显示器
        - transmissive: 透射式
        - reflective: 反射式

Graphics software

OpenGL

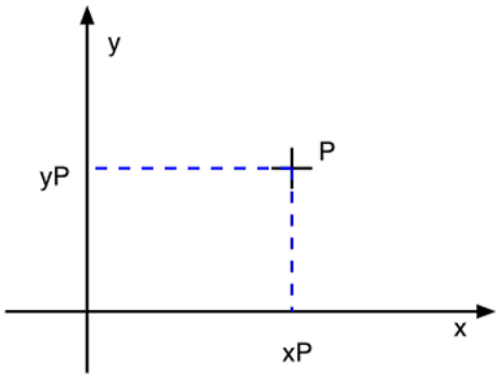
- OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms
- Similarly, OpenGL does not provide high-level commands for describing models of 3-dimensional objects
- You must build up your desired model from a small set of geometric primitives -points, lines, and polygons

Lecture 2 Mathematics for Computer Graphics

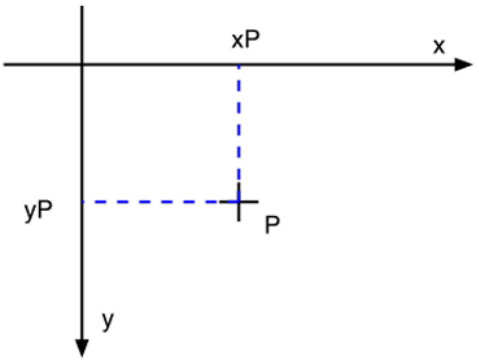
Computer representation of objects

3D points of objects $\rightarrow$ Manipulation of 3D points $\rightarrow$ Display of points on 2D screen
Model of 3D world $\rightarrow$ Transformation $\rightarrow$ Projection
Storage $\rightarrow$ Calculation $\rightarrow$ Display

Cartesian co-ordinate system



Representation of point
$P(x_p, y_p)$ in Cartesian co-ordinates



Representation of point
$P(x_p, y_p)$ on computer screen

Gradient of a line

- Gradient = $\Delta y / \Delta x$

Perpendicular lines

- The product of the slopes of two perpendicular lines is -1

Angles and trigonometry

- sin(θ) = Opposite / Hypotenuse
- consine(θ) = Adjacent / Hypotenuse
- tangent(θ) = Opposite / Adajacent

Vector

- Vector addition
- Vector subtraction
- Vector scaling
- Dot product of two vectors
  - $V_1 \cdot V_2 = |V_1||V_2|cos(\alpha)$
- Cross product of two vectors
  - $V_1 \times V_2 = |V_1||V_2|sin(\alpha)n$
    where 0 $\leq \alpha \leq$ 180 and n is a unit vector along
    the direction of the plane normal obeying the right hand rule
  - $V_1 \times V_2$ = -$V_2 \times V_1$

Matrices

- Matrices are techniques for applying transformattions
- A matrix is simply a set of numbers arranged in a rectangular format
- Each number is known as an element
- Capital letters are used to represent matrices, bold letters when printed **M**, or underlined when written $\underline{M}$.
- Dimensions of matrices: row $\times$ column
- Transpose matrix
    - When a matrix is rewritten so that its rows and columns are interchanged, then the resulting matrix is called the transpose of the original
- Square and symmetric matrices
    - A square matrix is a matrix where the number of rows equals the number of columns
    - A symmetric matrix is a square matrix where the rows and columns are such that its transpose is the same as the original matrix, i.e.,elements $a_{ij} = a_{ji}$ where i $\neq$ j
- Identity matrices
    - An identity matrix, I is a square and symmetric matrix with zeros everywhere except its diagonal elements which have a value of 1
- Non-commutative property of matrix multiplication
    - Matrix multiplication is not commutative
    - Reversing the order of the matrices yields different results
- Inverse matrices
    - If two matrices A and B, when multiplied together, results in an identity matrix I, then matrix A is the inverse of matrix B and vice versa, i.e.
        - A $\times$ B = B $\times$ A = I
        - A = B^{-1} and B = A^{-1}
        - $$\begin{bmatrix} 7 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & -3 \\ -2 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
            - $$A^{-1} = \begin{bmatrix} 1 & -3 \\ -2 & 7 \end{bmatrix}$$

Lecture 3 Geometric Primitives

Geometric Primitives

- Line characterizations
    - Implicit

$$y = mx + b$$

or

$$F(x,y) = ax + by + c = 0$$

- Parametric(explicit)

$$P(t) = (1-t)P_0 + tP_1$$

where

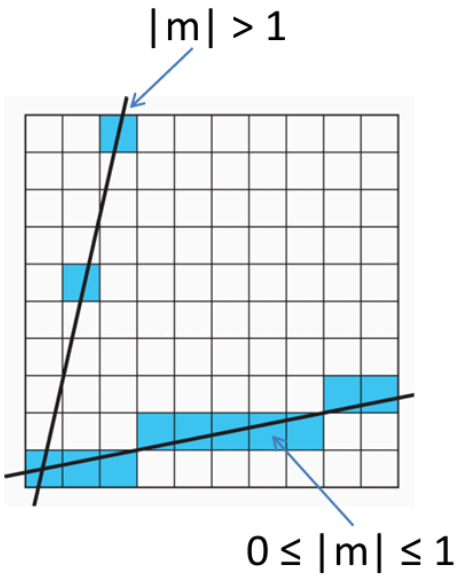
$$P(0) = P_0;\ P(1) = P_1$$

  - intersection of 2 planes
  - Shortest path between 2 points
  - convex hull of 2 discrete points

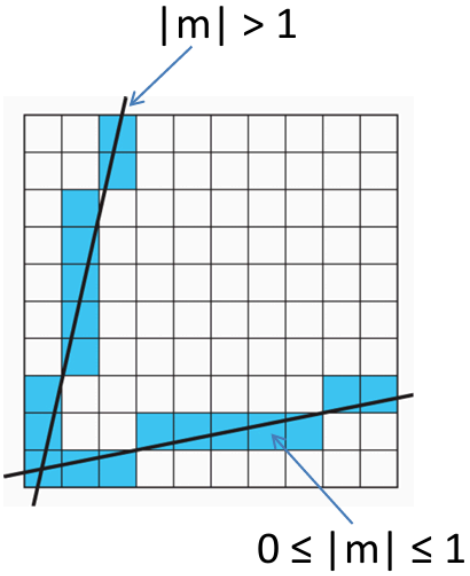
- Good discrete lines
  - No gaps in adjacent pixels
  - Pixels close to ideal line
  - Consistent choices; same pixels in same situations
  - Smooth looking
  - Even brightness in all orientations
  - Same line for P_0P_1 as for P_1P_0
  - Double pixels stacked up?

Line algorithms

- DDA-Digital Different Algorithm
  $y = mx + b$
  $m = (y_2 - y_1)/(x_2 - x_1) = \Delta y/\Delta x$
  $\Delta y = m\Delta x$
  As we move along x by incrementing x, $\Delta x = 1$, so $\Delta y = m$

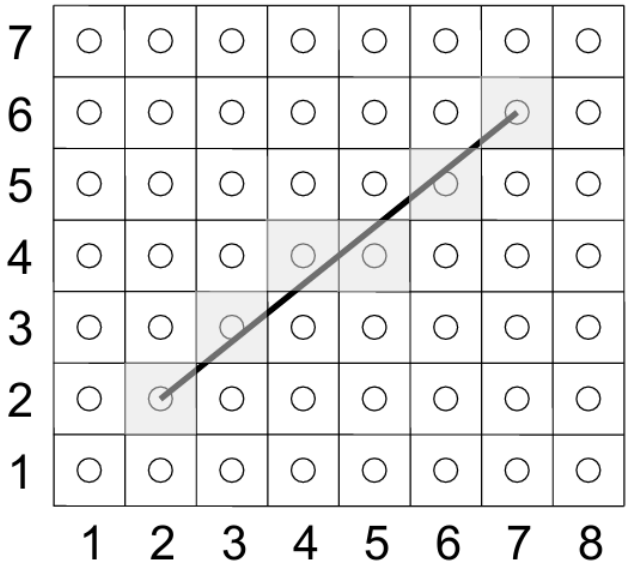


Sampling along x-axis for both lines



Sampling along x-axis (0≤|m|≤1)
Sampling along y-axis (|m|>1)

Always sampling along the major Axis, which changes faster



$$m = \Delta y / \Delta x$$
$$= (6-2) / (7-2)$$
$$= 0.8$$
$$y_{k+1} = y_k + m$$

x_{int}	y_{flot}	y_{int}
2	2.0	2
3	2.8	3
4	3.6	4
5	4.4	4
6	5.2	5
7	6.0	6

- The Bresenham line algorithm
  - is another incremental scan conversion algorithm
  - It is accurate and efficient
  - Its big advantage is that it uses only integer calculations(unlike DDA which requires float-point additions)
  - The calculation of each successive pixel requires only an addition and a sign test
  - It is so efficient that it has been incorporated as a single instruction on graphics chips

- Generation of circles
    - In Cartesian co-ordinate

$$(x-x_c)^2+(y-y_c)^2=r^2$$

    - The position of points on the circle circumference can be calculated by stepping along the x axis in unit steps from x_c-r to x_c+r and calculating the corresponding y value at each position as

$$y=y_c\pm\sqrt{r^2-(x-x_c)^2}$$

    - Considerable amount of computation
    - The spacing between plotted pixel positions is not uniform
        - This could be adjusted by interchanging x and y (stepping through y values and calculating the x values) whenever the absolute value of the slope of the circle is greater than 1.
          这可以通过在圆的斜率大于1时交换x和y（遍历y值并计算x值）来调整
    - In polar co-ordinates

$$\begin{cases}x=x_c+r\cos\theta\\y=y_c+r\sin\theta\end{cases}$$

        - when a display is generated with these equations using a fixed angular step size, a circle is plotted with equally spaced points along the circumference
        - To reduce calculations, a large angular separation can be used between points along the circumference and connect the point with straight-line segments to approximate the circle path
        - To a more continuous boundary on a raster display, the angular step size can be set at 1/r. This plots pixel positions that are approximately one unit apart
        - 考虑圆的对称性，可以减少计算量
            - considering symmetry conditions between octants, a point at position (x,y) on one-eighth circle sector is mapped into the seven circle points in the other seven octants of the $x-y$ plane (only calculating the points within $x=r$ to $x=y$)
        - More efficient circle algorithms are based on incremental calculation of decision parameters, which involves only simple integer operations
- Antialiasing by area averaging
    - Color multiple pixels for each x depending on coverage by ideal line

Polygons and triangles

- The basic graphics primitives are points, lines and polygons

    - A polygon can be defined by an order set of vertices
- Graphics hardware is optimized for processing points and flat polygons
- Complex objects are eventually divided into triangular polygons(a process called tessellation细分)
- Scan conversion - rasterization(将几何图形变成屏幕上的像素点)
    - The output of the rasterizer is a set of potential pixels(fragments) for each primitive, which carry informations for color and location in the frame buffer, and depth information in the depth buffer for further processing
      光栅化器的输出是每个图元的一组潜在像素（fragments 片元），这些潜在像素携带了帧缓冲器中的颜色和位置信息，以及深度缓冲器中的深度信息，用于进一步处理
    - The 3D to 2D projection gives us 2D vertices(points) to define 2D graphic primitives
    - We need to fill the interior
        - the rasterizer must determine which pixels in the frame buffer are inside the polygon
- Polygon fill
    - Polygon fill is a sorting problem, where we sort all the pixels in the frame buffer in those that are inside the polygon and those that are not
    - Rasterize edges into frame buffer
    - Find a seed pixel inside the polygon
    - Visit neighbors recursively and color if they are not edge pixels 递归的访问邻居，如果他们不是边像素则着色
    - There are different polygon-fill algorithms
        - Flood fill
        - ScanLine fill
        - Odd - even fill

Geometric primitives in OpenGL

- `glBegin(parameters)`
    - `GL_POINTS` : individual points
    - `GL_LINES` : pairs of vertices interpreted as individual line segments
    - `GL_LINE_STRIP` : series of connected line segments 一系列连接的线段
    - `GL_LINE_LOOP` : same as above, with a segment added between last and first vertices
    - `GL_TRIANGLES` : triples of vertices interpreted as triangles
    - `GL_TRIANGLE_STRIP` : linked strip of triangles 画一串相连的三角形带
    - `GL_TRIANGLE_FAN` : linked fan of triangles

  - `GL_QUADS` : quadruples of vertices interpreted as four-sided polygons
  - `GL_QUAD_STRIP` : linked strip of quadrilaterals
  - `GL_POLYGON` : boundary of a simple, convex polygon
- `glVertex2f()` : create new vertex

Lecture 4 Transformation Pipeline and Geometric Transformations

Transformation Pipeline

- The Transformation Pipeline is the series of transformations(alterations) that must be applied to an object before it can be properly displayed on the screen
- The transformations can be thought of as a set of processing stages
  - If a stage is omitted, very often the object will not look correct
  - For example if the projection stage is skipped then the object will not appear to have any depth to it
- Once an object has passed through the pipeline it is ready to be displayed as either a wire-frame item or as a solid item
- 5 transformation
  - Modeling Transformation - to place an object into the Virtual World
  - Viewing Transformation - to view the object from a different vantage point in the virtual world 从虚拟世界的各种不同角度看世界
  - Projection Transformation - to see depth in the object
  - Viewport Transformation - to temporarily map the volume defined by the "window of interest" plus the front and rear clipping planes into a unit cube. When this is the case, certain other operations are easier to perform
  - Device Transformation - to map the user defined "window of interest"(in the virtual world) to the dimensions of the display area
- We start when the object is loaded from a file and is ready to be processed
  从对象从文件加载并准备好处理时开始
- We finish when the object is ready to be displayed on the computer screen
  从对象准备好显示在计算机屏幕上时结束
- We should be able to draw a picture of a simple object, say a cuboid, and show visually what happens to it as it passes through each pipeline stage

Standard transformation

2D translation

- Translating a point from $P(x, y)$ to $P'(x', y')$ along vector T

- Importance in computer graphics - we need to only transform the two endpoints of a line segment and let the implementation draw the line segment between the transformed endpoints

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

where $P(x, y)$ and $P'(x', y')$ are the original and new positions, and T is the distance translated

$$\mathbf{P}' = \mathbf{P} + \mathbf{T} \qquad \text{or} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

2D rotation

- Rotating a point from $P(x, y)$ to $P'(x, y)$ about the origin by angle $\theta - radius$ stays the same, and the angle increases by θ

$$\begin{cases} x' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ y' = r\sin(\phi + \theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \end{cases}$$

$$\begin{cases} x = r\cos\phi \\ y = r\sin\phi \end{cases}$$

$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases}$$

$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$$

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad \text{so} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

where θ is the rotation angle and ϕ is the angle between the x-axis and the line from the origin to (x, y). Notice that the rotation point(or pivot point) is the co-ordinate origin

- Rotation about a fixed point rather than the origin
  - Move the fixed point to the origin
  - Rotate the object
  - Move the fixed point back to its initial position
  - $M = T(p_f)R(\theta)T(-p_f)$

2D scaling

- When an object is scaled, both the size and location change
- To scale at a fixed point rather than the origin, the same process applies as for rotating about a fixed point

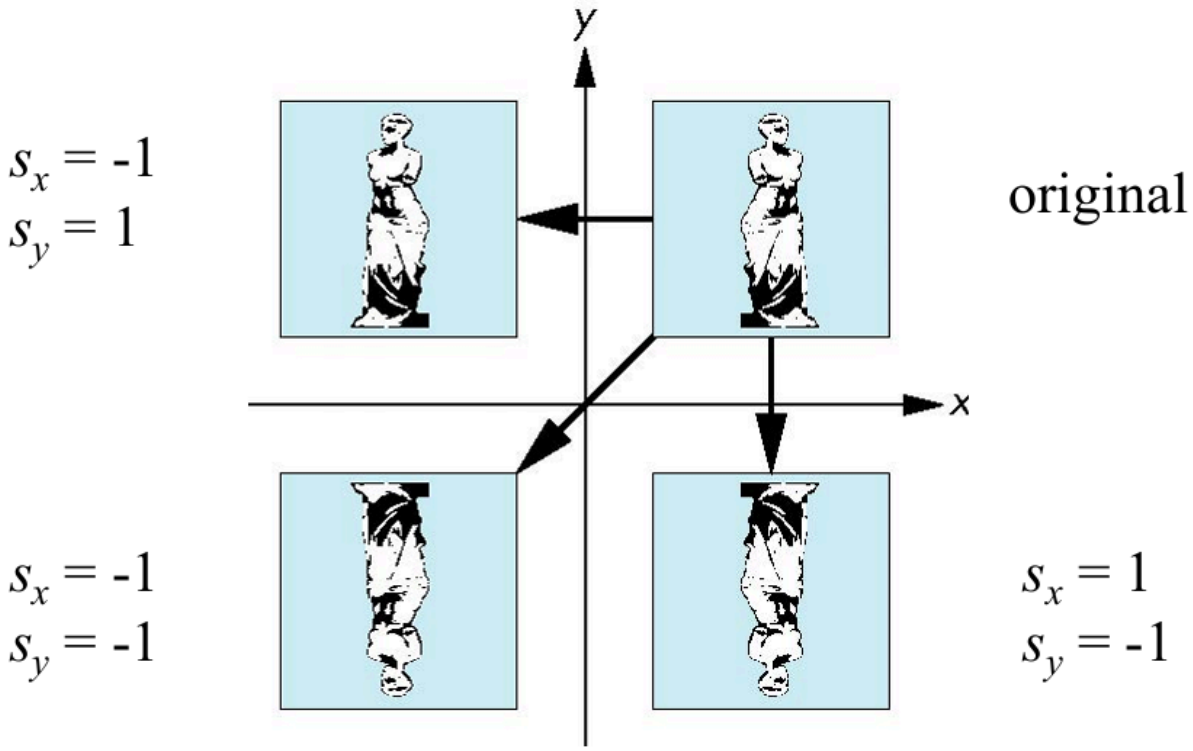
$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

so $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$

where P, and P' are the original and new positions, and s_x and s_y are the scaling factors along the $x-$ and $y-$ axes

2D reflection

- Special case of scaling - corresponding to negative scale factors



2D shearing

- Equivalent to pulling faces in opposite directions
- $x' = x + y \cot \theta$, $y' = y$

2D homogeneous co-ordinates

- By expanding the 2×2 matrices to 3×3 matrices, homogeneous co-ordinates are used
- A homogeneous parameter is applied so that each position is represented with homogeneous co-ordinates(h_x, h_y, h)
- The homogeneous parameter has a non zero value, and can be set to 1 for convenient use
- This makes all the transformation matrices into the same format, i.e. 3×3 matrices, including the translation matrix
- The homogenous transformation matrices can then be combined into a composite transformation matrix
- $(x, y) \rightarrow (x, y, 1)$

- 平移矩阵：

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

- 旋转矩阵：

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- 缩放矩阵：

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- 错切矩阵：

$$\begin{bmatrix} 1 & \cot\theta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D composite transformation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} rs_{xx} & rs_{xy} & trs_x \\ rs_{yx} & rs_{yy} & trs_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where elements rs are the multiplicative rotation-scaling terms in the transformation(which involve only rotation angles and scaling factors)
elements trs are the translation terms, containing combination of translation distances, pivot-point and fixed-point co-ordinates, rotation angles and scaling
平移距离 枢纽点 固定点坐标 旋转角度 缩放参数

3D transformation

3D translation

- 3D translation and scaling can be simply extended from the corresponding 2D methods

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D co-ordinate axis rotations

- The extension from 2D rotation methods to 3D rotation is less straightforward(because this is about an arbitrary axis instead of an arbitrary point)
- Equivalent to rotation in 2 dimensions in planes of constant z(i.e. about the origin)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (\text{About z-axis})$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (\text{About x-axis})$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (\text{About y-axis})$$

- General rotation about the origin
  > - A rotation by q about an arbitrary axis can be decomposed into the concatenation of rotations about the x, y, and z axes
  >
  > $$R(q) = R_z(q_z)R_y(q_y)R_x(q_x)$$
  >
  > where q_x,q_y and q_z are called the Euler angles. Note that rotations do not commute though we can use rotations in another order but with different angles

3D composite transformation

- As with 2D transformation, a composite 3D transformation can be formed by multiplying the matrix representations for the individual operations in the transformation sequence
- There are further forms of transformation, namely reflection and shearing which can be implemented with the other three transformations
- Translation, scaling, rotation, reflection and shearing are all affine transformations in that transformed point $P'(x', y', z')$ is linear combination of the original point $P(x, y, z)$
- Matrix multiplication is associative
  $M_3 \cdot M_2 \cdot M_1 = (M_3 \cdot M_2) \cdot M_1 = M_3 \cdot (M_2 \cdot M_1)$
  but transformation products are not always commutative

OpenGL

OpenGL matrices

- OpenGL is a state machine
- Modes and attributes in OpenGL remain in effect until they are changed

- Most state variables can be enabled or disabled with `glEnable()` or `glDisable`
- In OpenGL, matrices are part of the state
- Multiple types
    - Model-view (`GL_MODELVIEW`)
    - Projection(`GL_PROJECTION`)
    - Texture(`GL_TEXTURE`)
    - Color(`GL_COLOR`)
- Single set of functions for manipulation 单一的函数集进行操作
- Select which to be manipulated by
    - `glMatrixMode(GL_MODELVIEW)`
    - `glMatrixMode(GL_PROJECTION)`

Current transformation matrix

- Conceptually there is a 4×4 homogeneous coordinate matrix, the current transformation matrix(CTM) that is part of the state and is applied to all vertices that pass down the pipeline
- The CTM is defined in the user program and loaded into a transformation unit
- The CTM can be altered either by loading a new CTM or by post multiplication
- OpenGL has a model-view and a projection matrix in the pipeline which are concatenated together to form the CTM
- The CTM can manipulate each by first setting the correct matrix mode

Arbitrary Matrices

- We can load and multiply by matrices defined in the application program
    - `glLoadMatrixf(m)`
    - `glMultMatrixf(m)`
- Matrix m is one-dimension array of 16 elements which are the components of the desired 4×4 matrix stored by columns
- In `glMultMatrixf` , m multiplies the existing matrix on the right

Matrix stacks

- CTM is not just one matrix but a matrix stack with the "current" at top
- In many situations we want to save transformation matrices for use later
    - Traversing hierarchical data structures
    - Avoiding state changes when executing display lists
- we can also access matrices with query functions
    - `glGetIntegerv`

> - `glGetBooleanv`
> - `glGetFloatv`
> - `glGetDoublev`
> - `glIsEnable`

OpenGL transformation function

- `glTranslate`: specify translation parameters
- `glRotate`: Specify rotation parameters for rotation about any axis through the origin
- `glScale`:Specify scaling parameters with respect to the co-ordinate
- `glMatrixMode`: Specify current matrix for geometric-viewing, projection, texture or color transformation
- `glLoadIdentity`: Set current matrix to identity
- `glLoadMatrix`: Set elements of current matrix
- `glMultMatrix`: Post-multiply the current matrix by the specified matrix
- `glGetIntegerv`: Get max stack depth or current number of matrices in the stack for selected matrix mode
- `glPushMatrix`: copy the cop matrix in the stack and store copy in the second stack position
- `glPopMatrix()`: Erase top matrix and move second matrix to top stack

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0,2.0,3.0);
glRotatef(30.0,0.0,0.0,1.0)
glTranslatef(-1.0,-2.0,-3.0)
```

先告诉OpenGL接下来操作的是模型视图矩阵，然后重置当前矩阵为单位矩阵（把之前的旋转移动缩放都清空），然后对于一个点先执行（`glTranslatef(-1.0,-2.0,-3.0)`）把物体移动，使得你想围绕旋转的那个中心点`(1,2,3)`被移动到坐标原点，然后再执行旋转第一个参数是角度，然后是定义旋转轴，旋转按照右手定则，逆时针旋转，最后旋转完了再把物体一回原来的位置。
这里不用`glPushMatrix/glPopMatrix`是应为这里已经通过loadidentity暴力重置了，如果要画一个太阳系，画完太阳画地球（在太阳旁边，需要自转），如果要画火星，需要使用pop恢复存档，火星会基于地球的坐标系继续变换

Lecture 5 Viewing and Projection

Classic viewing

Viewing 观察/视图需要三个基本元素

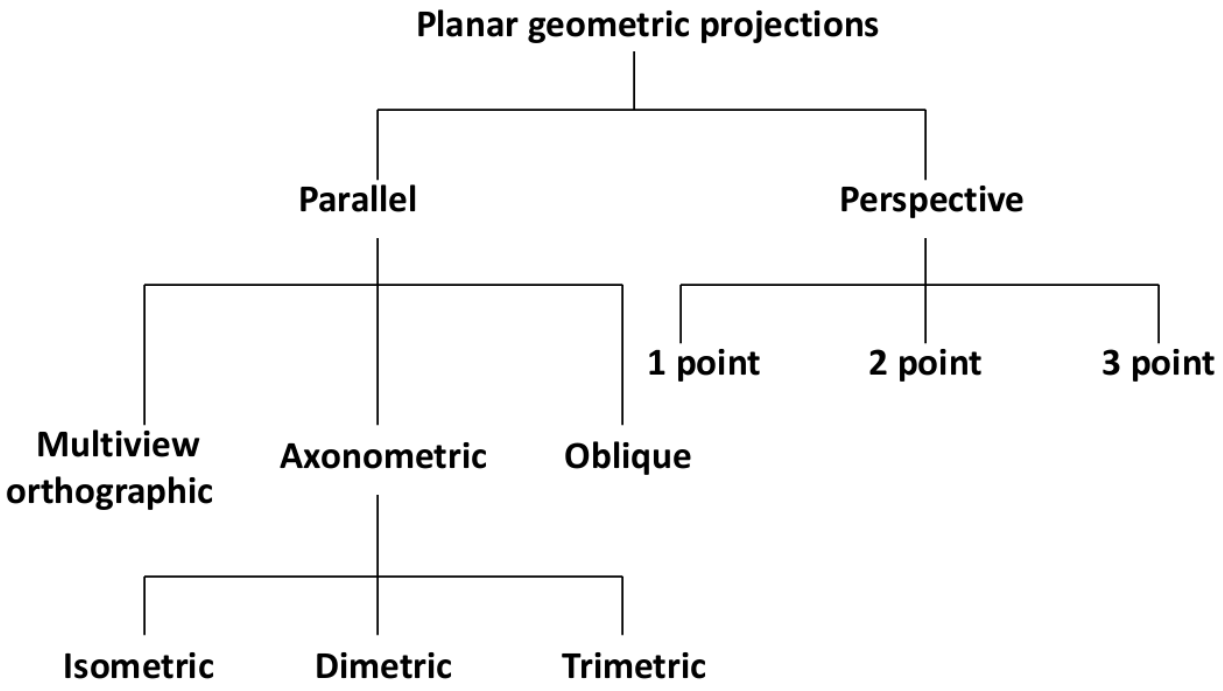
- one or more objects一个或多个对象
- a viewer with a projection surface 一个具有投影平面的观察者
- Projectors that go from the object to the projection surface 从物体到投影平面的投影
  Classical views are based on the relationship among three elements
  - the viewer picks up the object and orients the object in way that it is to be seen
- Each object is constructed from flat principle faces
  - buildings, polyhedra, manufactured objects
- there are 3 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

Types of projection and their advantages / disadvantages

Planar geometric projection(平面几何投影)

- Standard projection project onto a plane
- Projectors are lines that either coverage at a center of projection or are parallel 对于投影线可以是从一个点发散出去的，也可以是平行的
- 对于投影线从一个中心点发散出去，而这些投影线会汇聚到中心投影点的这种投影方式叫Parallel（中心投影）或者 Perspective（透视投影）
- Such projections preserve lines but not necessarily angles 平面几何投影可以保存物体上的直线，但不一定能保存直线之间的夹角
- Non-planar projections are needed for applications such as map construction 非平面几何投影在一些场景时必要的，比如地图制作中

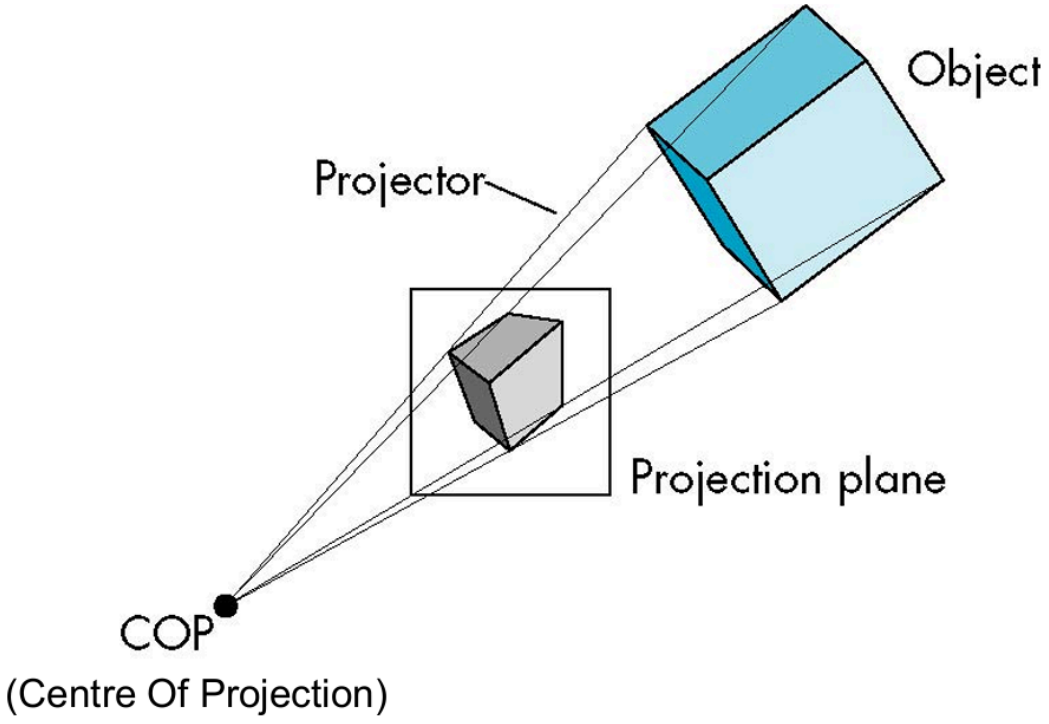
Taxonomy of planar geometric projection



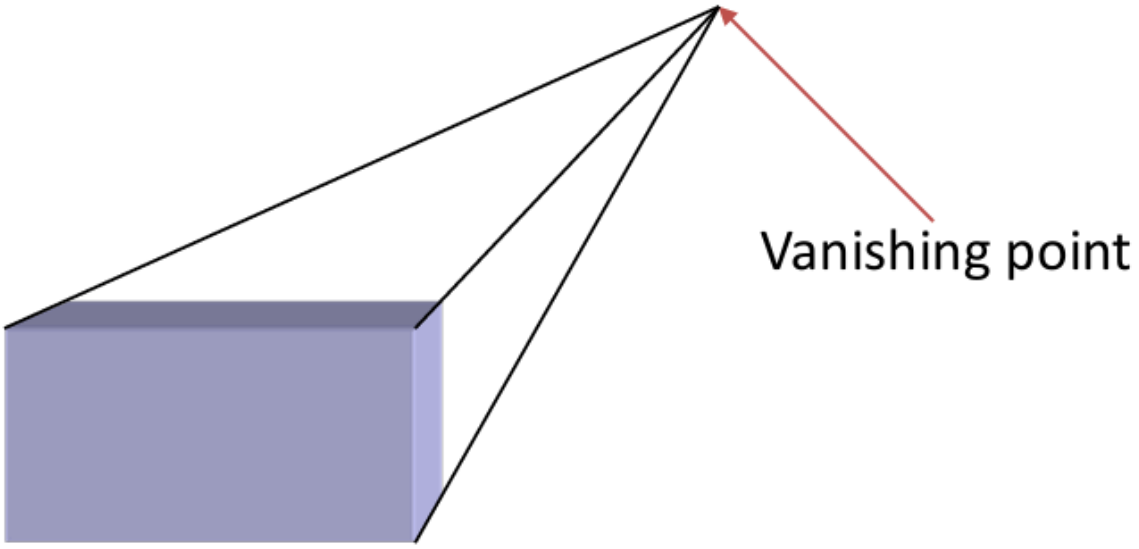
- 在计算机图形学中，所有的投影都以相同的方式处理，并使用单一的管道来实现它们
- 在经典的视图方式中，针对每种类型的投影都发展了不同的绘制技术
- 基本区别在于透视视图和平行视图，尽管在数学上，平行视图是透视视图的极限情况

Perspective projection 透视投影

- Perspective projection generates a view of 3-dimensional scene by projecting points to the view plane long converging paths 通过沿汇聚路径将点投影到视图平面来生成三维场景



- 符合现实中的近大远小，更逼真
- Vanishing points 消失点



  - Parallel lines(not those parallel to the projection plane) on the object converge at a single point in the projection
  - Drawing simple perspectives by hand uses the vanishing points
- Three-point perspective

  - No principle face parallel to the projection plane
  - Three vanishing points for the cube

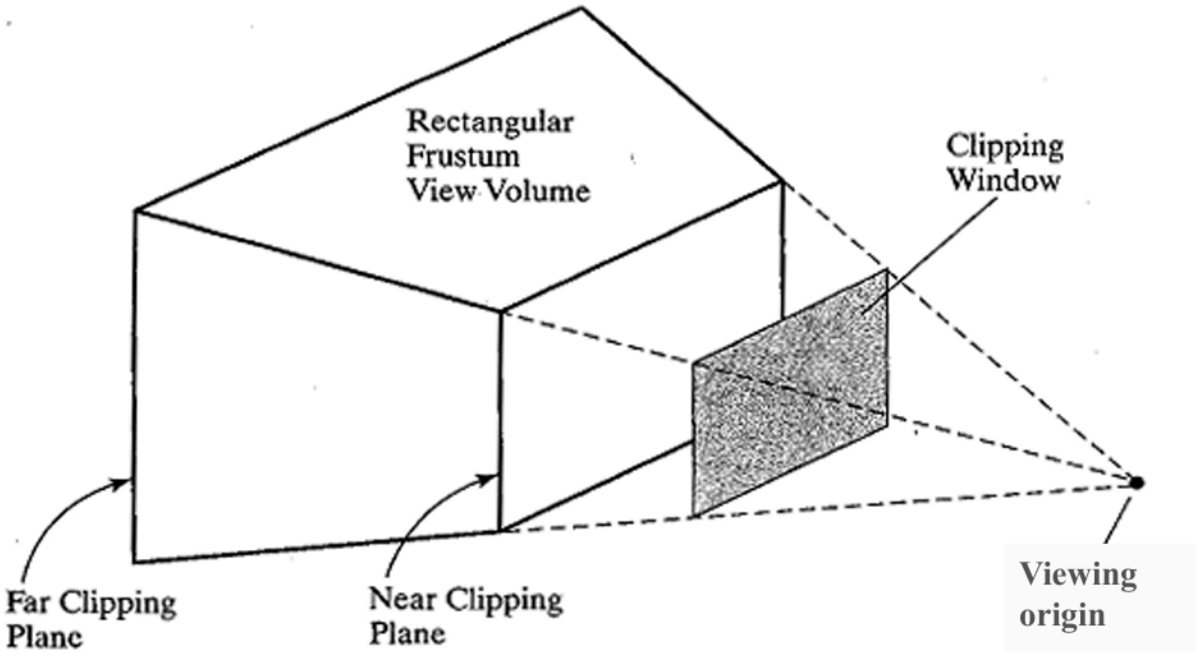
- Two-point perspective

  - One principal direction parallel to the projection plane
  - Two vanishing points for the cube
- One-point perspective

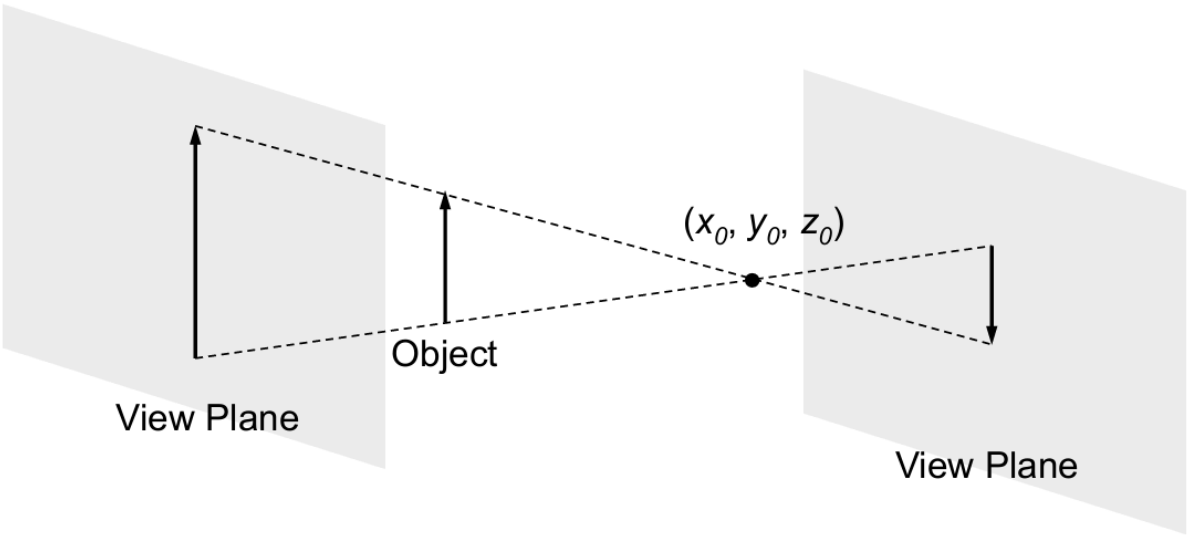
  - One principal face parallel to the projection plane

    - One vanishing point for the cube
- Advantages
    - objects further from the viewer are projected smaller than the same sized objects closer to the viewer 距离观察者较远的物体比距离较近的同尺寸物体投影得更小（衰减）
    - Angles preserved only in planes parallel to the projection plane
- Disadvantages
    - Equal distances along a line are not projected into equal distances
    - more difficult to construct by hand than parallel projections(but not more difficult by computer)

Frustum perspective projection(视锥透视投影)



- By adding near and far clipping planes that are parallel to the viewing plane, pats of the infinite perspective view volume are chopped off to form a truncated pyramid or frustum 通过添加平行于观察品面的近裁剪平面和原裁剪平面，无限透视视景体的部分被切掉
- Some systems(OpenGL 规定眼睛→屏幕→物体) restrict the placement of the viewing plane relative to the near and far planes, but for some system these clipping plane is choosable
- 近裁切面可用于去除靠近观察平面的大型物体，
- 当观察点距离观察平面非常远时，透视投影趋进于平行投影
- 如果观察平面在观察点后方，物体会在观察平面上导致，如果观察平面在物体后方，物体在投影时只会被放大

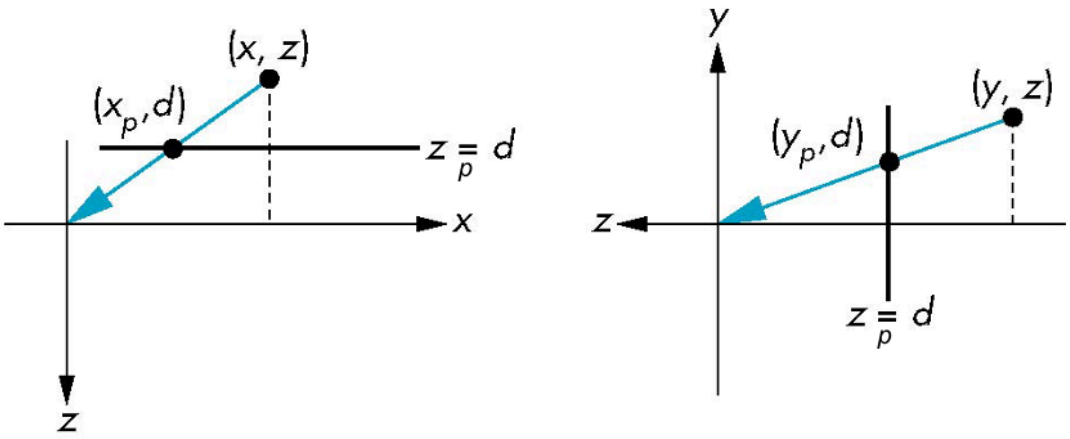


Objects are enlarged if the viewing plane is behind the objects.

Objects are inverted if the viewing plane is behind the projection reference point.

Simple perspective projection

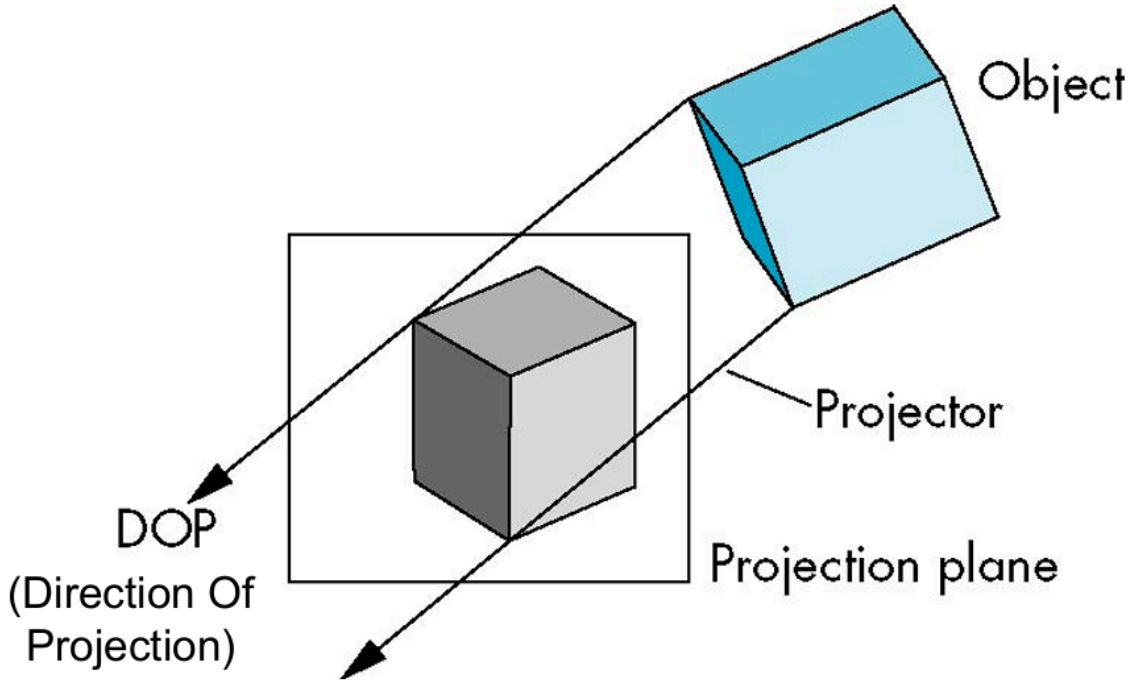
- Centre of Projection(CoP) at the origin
- Projection plane $z = d, d < 0$
- consider top and side views



- $x_p = \frac{x}{z/d}\ y_p = \frac{y}{z/d}\ z_p = d$

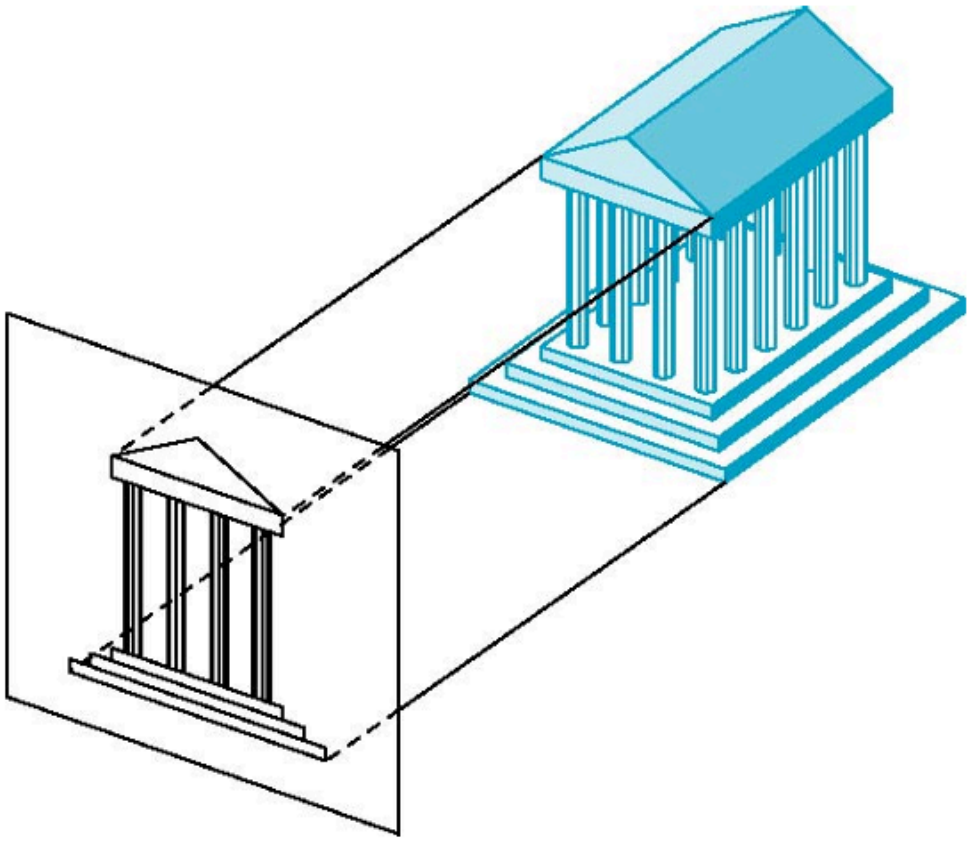
Parallel projection 平行投影

- This method projects points on the object surface along parallel lines 此方法沿平行线在对象平面投影点
- It is usually used in engineering and architecture drawings to represent an object with a set of views showing accurate dimensions



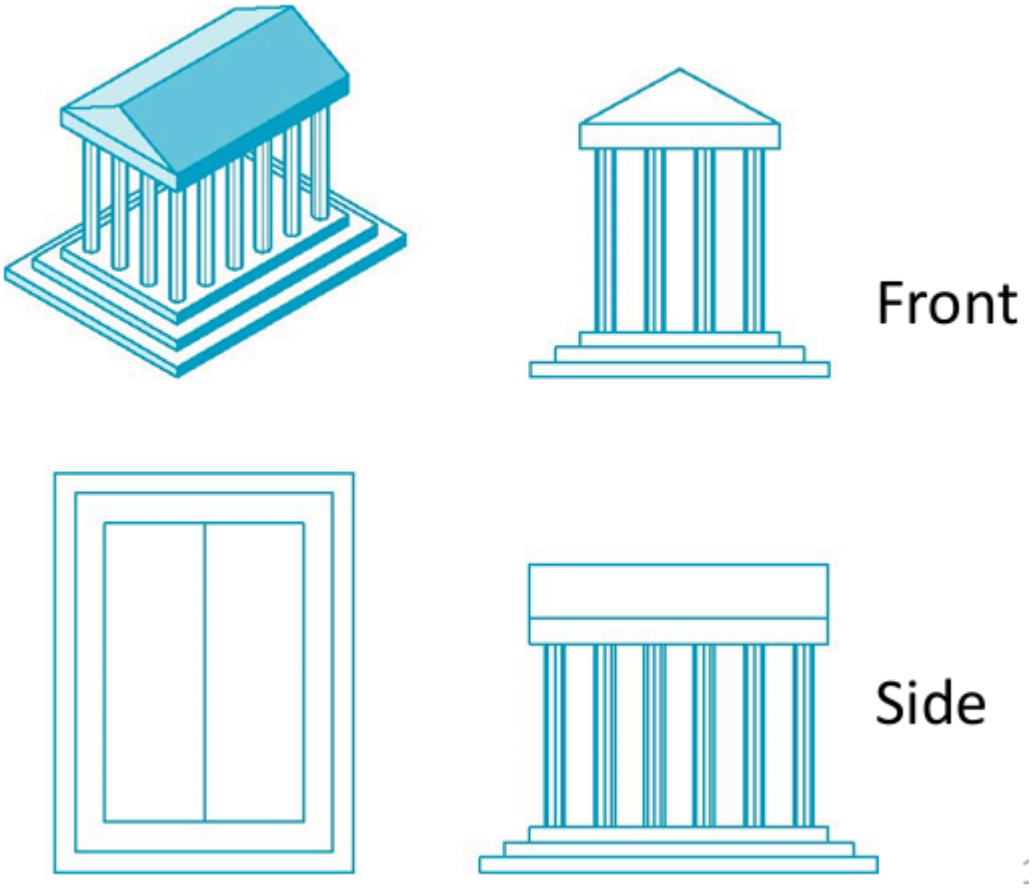
Orthographic projection(正投影)

投影线与投影平面正交



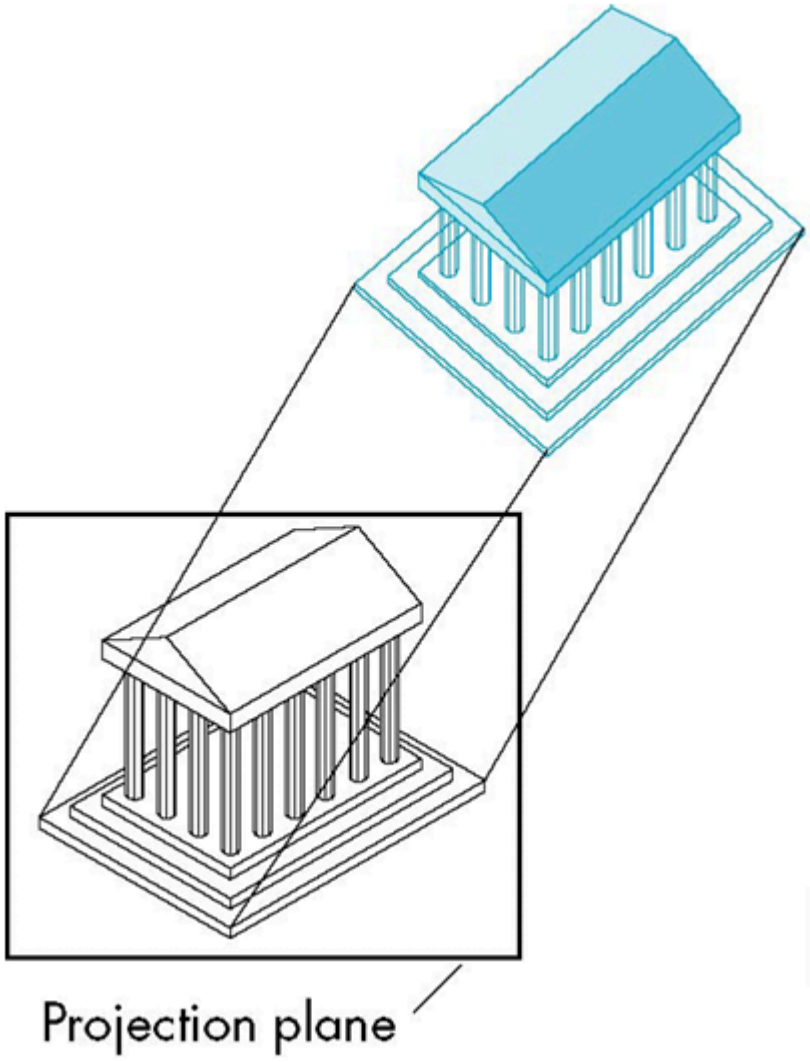
Multiview orthographic projection(多视图正投影)

投影平面平行于主面

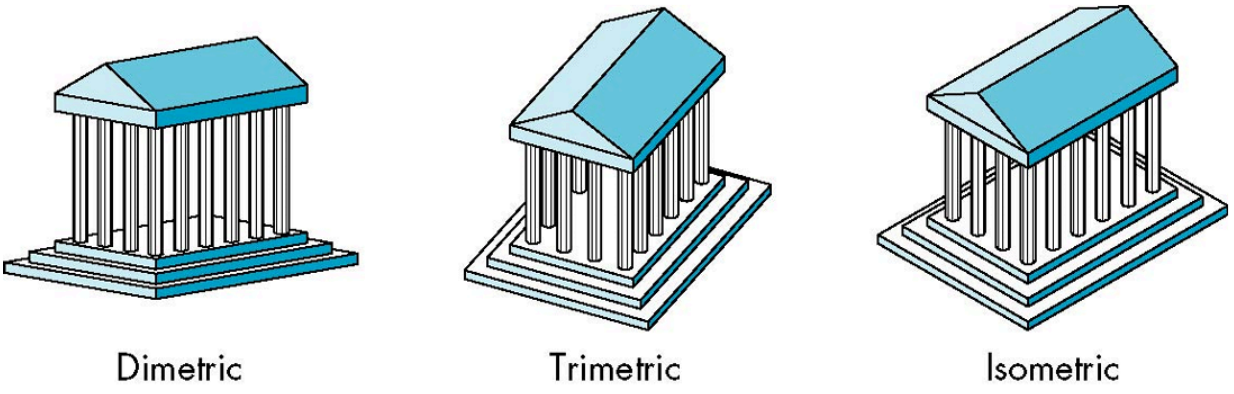


- Advantages:
  - preserves both distance and angles
    - shapes preserved
    - can be used for measurements
- disadvantages
  - cannot see what object really looks like because many surfaces are hidden from the view（因此我们经常添加一个Isometric view等轴测图）

Axonometric projections(轴测投影)



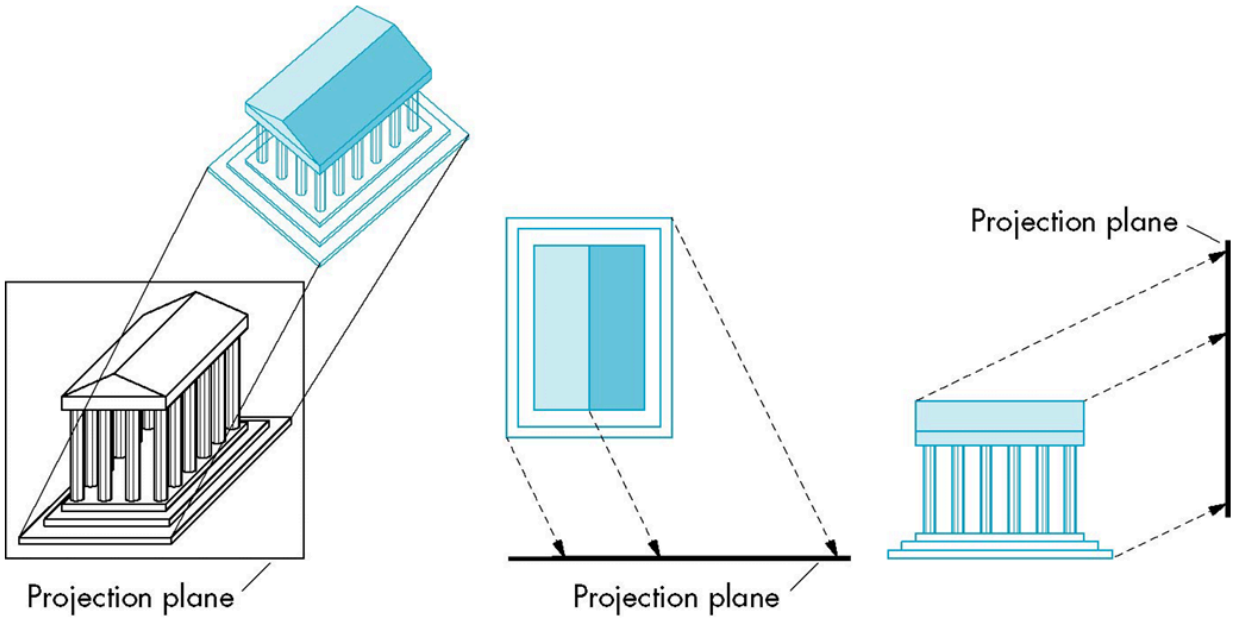
- allow projection to move relative to the object 允许投影平面相对于物体进行运动
- If the projection plane is places symmetrically with respect to the 3 principle faces that meet at a corner of our rectangular object, then we have an isometric view 如果投影平面相对于矩形物体一觉相交的三个主面对称放置，则得到等轴测图
- If the projection is placed symmetrically with respect to the 2 principle faces that meet at a corner of our rectangular object, then we have an dimetric view
- The general case is a trimetric view



- Advantages:
  - lines are scaled(foreshortened) but can find scaling factors

    - lines preserved but angles are not
    - can see three principal faces of a box-like object
- Disadvantages
    - Some optical illusions possible
        - parallel lines appear to diverge
    - Does not look real because far objects are scaled the same as near objects

Oblique projection(斜投影)

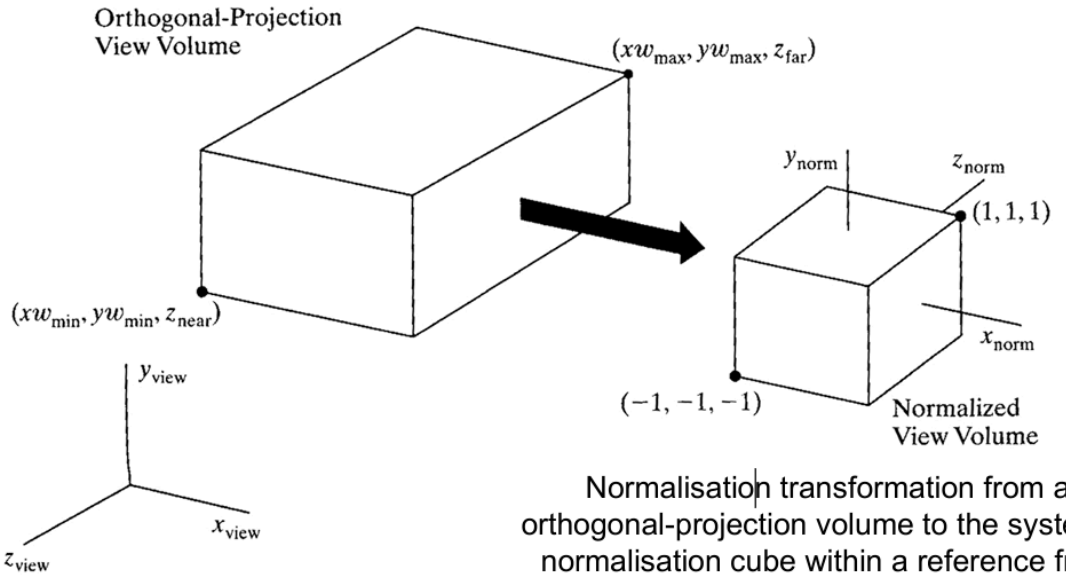


- Arbitrary relationships between projectors and projection plane 投影线与投影平面之间为任意关系
- Advantages:
    - can pick the angles to emphasis a particular face 可以选择角度以强调特定的面
        - Architecture: plan oblique, elevation oblique
    - Angles in faces parallel to the projection plane are preserved while we can still see around side 平行于投影平面的角度得以保留
    - In the physical world, we cannot create oblique projections with a simple camera; possible with bellows camera or special lens(architectural)

Orthogonal projection

- 从用户定义的视野映射到标注设备坐标的过程
- Orthogonal projection is a transformation of object descriptions to a view plane along lines parallel to the view-plane normal vector N 正交投影是将物体描述沿平行于平面法向量的直线变化到观察平面
- It is often used to produce the front, side and top views of an object
- Engineering and architectural drawings commonly employ these orthographic projections since the employ these orthographic projections since the lengths and angles are

accurately depicted and can be measured from the drawings

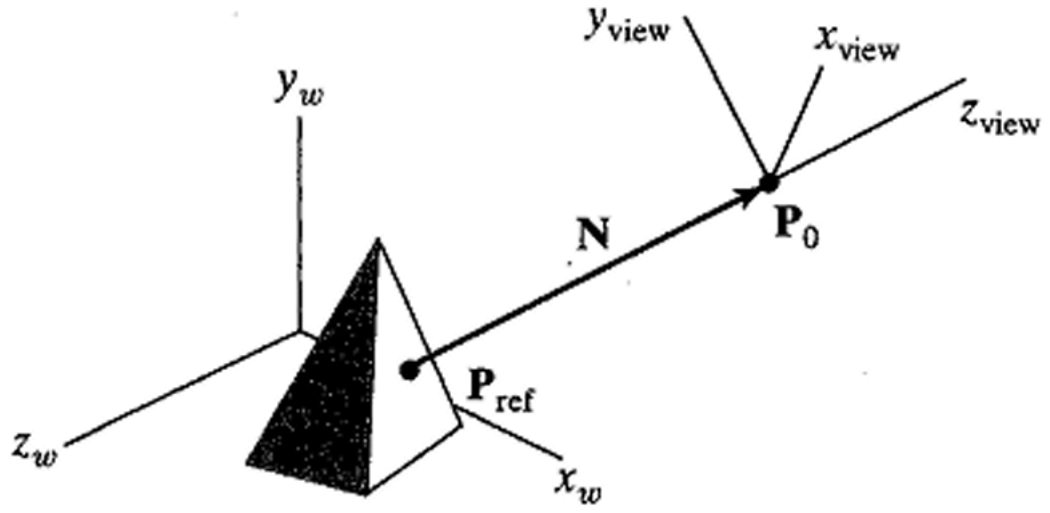


Normalisation transformation from an orthogonal-projection volume to the systematic normalisation cube within a reference frame

- 电脑通过把这个长方体平移到原点然后挤压或拉伸，原本平行的线变换后依然平行，物体不会出现近大远小的透视效果

3D viewing co-ordinate parameters

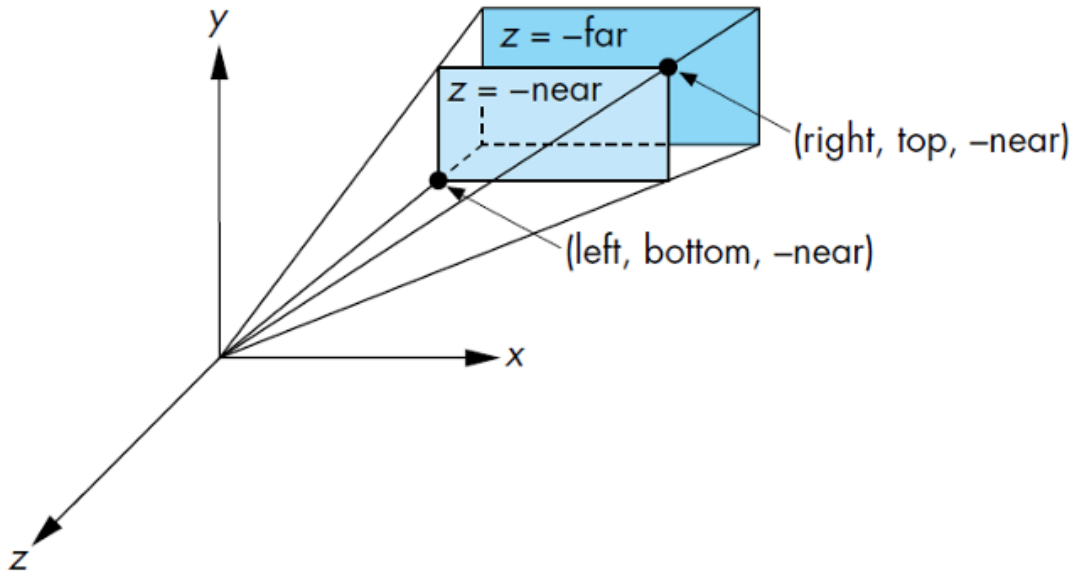
- A world co-ordinate position $P_0(x_0, y_0, z_0)$ is selected as the $viewing\ origin$ (also called viewing point, viewing position, eye position or camera)
- The direction from a reference point $P_{ref}(x_{ref}, y_{ref}, z_{ref})$ to the viewing origin $P_0(x_0, y_0, z_0)$ can be taken as the viewing direction(vector), and the reference point is termed the look-at point 从参考点到观察原点的方向可以取为观察方向，该参考点称为观察目标点
- The viewing plane can then be defined with a normal vector N, which is the viewing direction, usually the negative z_{view} axis 可以用法向量N定义观察平面，即观察方向，通常是负的z_{view}轴
- The viewing plane is thus parallel to the x_{view}-y_{view} plane



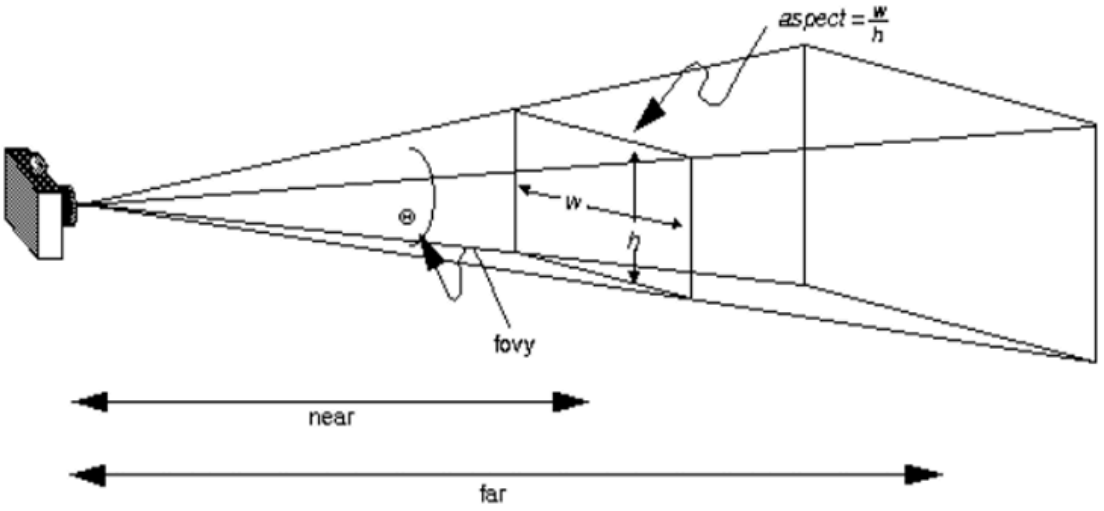
- 左边x_w, y_w, z_w世界坐标系，右边$x_{view}, y_{view}, z_{view}$ 是以摄像机为中心的坐标系，这个坐标系中摄像机是中心，N决定了相机z轴的方向（相机看向z轴的负方向）但完全定义一个相机，一般还需要一个向量Up Vector(V)，用来定义相机的头顶朝哪儿，他不一定完全等同于y_{view}，坐标系是通过正交化来求出来的，先通过法向量N确定z轴，然后再根据头顶向量与z轴的叉乘算出x轴，最后通过x轴叉乘N向量从而算出准确的y轴，严格垂直于视线

OpenGL functions

- `gluLookAt(eye_position,look_at,look_up`: Specify three dimensional viewing parameters
- `glOrtho(GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat near, GLfloat far)`:Specify parameters for a clipping window and the near and far clipping planes for an orthogonal projection 定义正交投影的裁剪窗口以及近裁剪平面和原裁剪平面参数
- `glFrustum(GLfloat fov, GLfloat aspect, GLfloat near, GLfloat far)`:指定透视投影的裁剪窗口以及近裁剪平面和远裁剪平面的参数



- `gluPerspective(GLfloat fov, GLfoat aspect, GLfloat near, GLfloat far)`: Specify field-of-view angle(fov which is a matrix) in the y-direction, aspect ratio of the near and far planes; it is less often used than `glFrustum`



`

Lecture 6 Parametric Curves and Surfaces

Why Parametric?

- parametric surfaces are surfaces that are usually parameterized with 2 independent variables
- By parameterization, it is relatively easy to represent surfaces that are self-intersecting or non-orientable(do not have well-defined normal vector direction everywhere)
- It is impossible to represent many of these surfaces by using implicit functions

Parametric curves

A curve in a 2D (x, y) surface is defined as :
$x = x(t)$
$y = y(t)$
In this way, the curve is well defined, each value of t in [0,1] defining one and only one point
Parametric equation of a straight line

- Explicit representation $y = a_0 + a_1 x$
- Parametric representation: $0 \leq t \leq 1$
  $x = x_1 + t(x_2 - x_1)$
  $y = y_1 + t(y_2 - y_1)$
  Parametric equation of a circle
- Implicit representation $x^2 + y^2 = r^2$
- Parametric equation:
  $x = r\cos(360t)$
  $y = r\sin(360t)$
  Parametric equation of a cubic curve
  $x = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

$y = b_0 + b_1 t + b_2 t^2 + b_3 t^3$

where a_1 and b_i terms are constants that vary from curve to curve

A polynomial can therefore be used

$x(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n$

For interpolation, if there are k points, then $n = k - 1$ must be chosen, in order to find the correct values for a_i

Low-degree polynomial curves

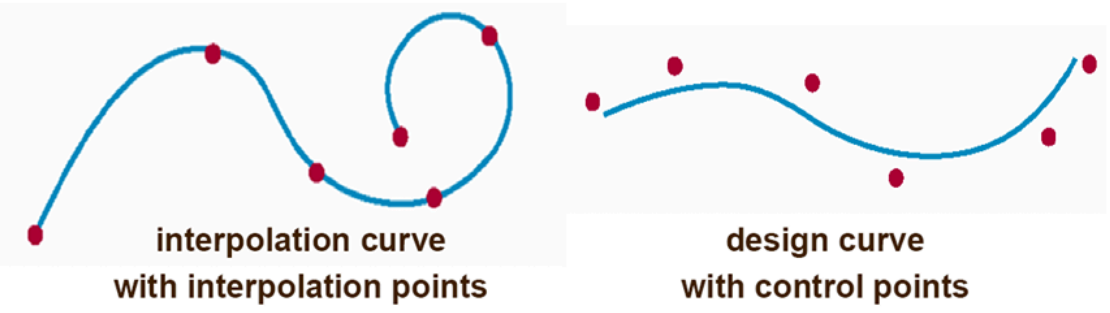
- Polynomials have to be used for efficiency
- High-degree polynomials are not suitable because of their behavior
- For curves of a large number of points
    - They can be broken into small sets(e.g. 4 points in each set)
    - a low-degree polynomial is put through each set
    - these individual curves(cubics) are joined up smoothly
- This is the basis of splines

Splines

- A splines consists of individual components, joined together smoothly, looking like a continuous smooth curve. 样条由单独的组成部分构成，平滑的连接在一起，看起来像一条连接平滑的曲线
- Different types of continuity exist and the following are generally required
    - continuity of the curve(no breaks)
    - continuity of tangent(no sharp conners)
    - continuity of curvature(not essential but avoids some artifact from lighting)
- 为了实现连续性，通常需要一个三次多项式来描述每个组成部分

Interpolation and design curves 插值与设计曲线

- 插值曲线定义了曲线必须穿过的精确位置 插值曲线的形状取决于提供的数据点
- 设计曲线定义了曲线的一般行为：曲线应该是什么样子，并且通常需要调整形状 设计曲线的形状取决于控制点，这些点通常不在曲线上，但是允许通过移动这些点来调整形状



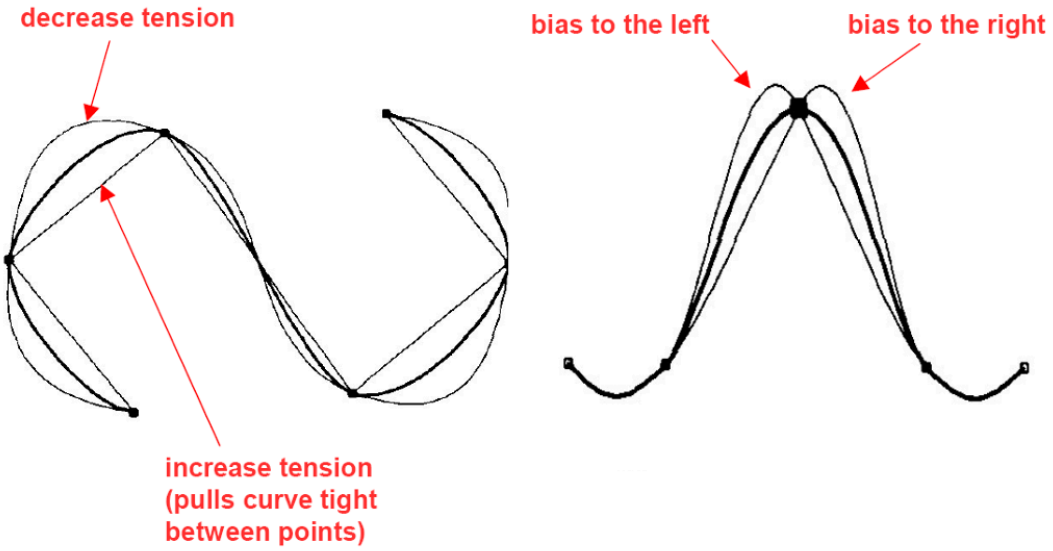
- An important feature of design curves is local control. When a curve is designed, if one part is done, it could be preferred to keep its current shape when another part of the curve is adjusted

Linear interpolation

Linear interpolation is useful(e.g. for animation) where t varies from 0 to 1 over time

Local control

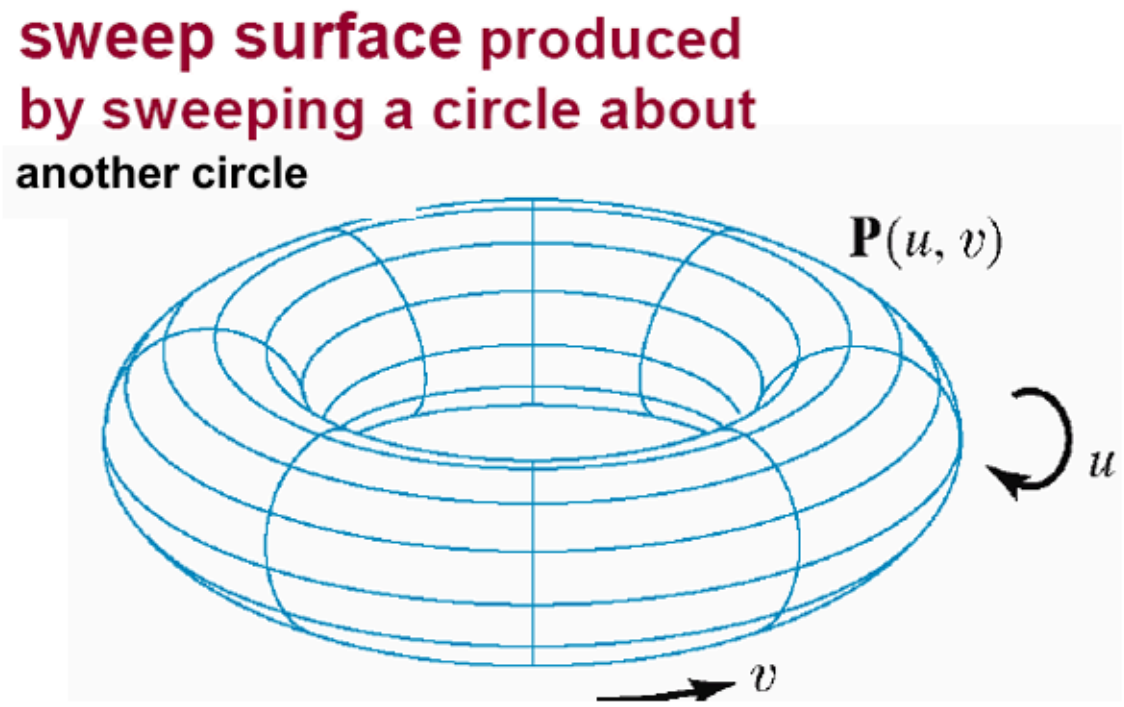
- Curves without local control
  - Natural splines
  - Bezier curves(if continuity enforced)
- Curves with local control
  - B-Splines
  - NURBS(Non-uniform Rational B-Splines)
- A cubic curve with local control 具有局部控制的三次曲线
  - Normally influenced by only 4 control points, which are the control points most local to it
- Forms of local control
  - control points(considered before)
  - Tension 张力
  - Bias 偏差



Types of parametric surface

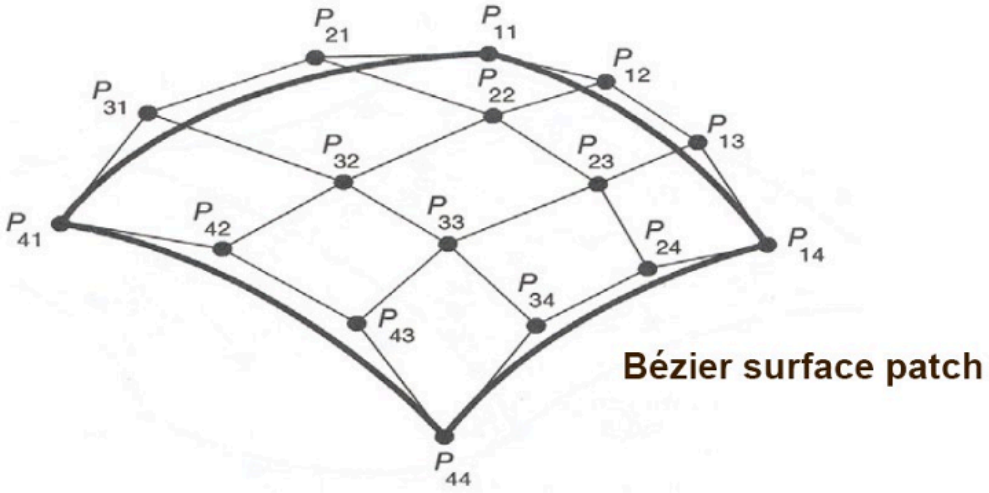
- Revolved surface - a 2D curve is revolved around an axis, and the parameter is the rotation angle 旋转曲面

- Extruded surface - a 2D curve is moved perpendicular to its own plane, and the parameter is the straight-line depth 拉伸曲面 一条2D的曲线沿着垂直于其自身平面的方向移动，参数是直线深度
- Swept surface - a 2D curve is passed along a 3D path(which can be a curve), and the parameter is the path definition 扫掠平面 一条2D曲线沿着3D路径移动，参数是路径定义



- Tensor product surfaces
  - Tensor product surfaces are the most widely used parametric surfaces
  - Parametric curves depend on 1 parameter(t) while parametric surfaces depend on 2 parameter (u, v)
  - A tensor product surface combines two parametric curves
  - Essentially these work in perpendicular direction
- Interpolation and design surfaces
  - As for curves, there are interpolation and design surfaces, and the design form is more common
  - There is a control grid, normally a rectangular array of control points 存在一个控制网络通常是控制点的矩阵阵列
  - As the curve is broken into smaller curves, the surface is broken into surface patches
  - Local control important

- Control grid for a surface patch



  - 在局部控制的三次曲线中，一个曲线段通常由4个控制点影响，在局部控制的三次曲面中，一个曲面片受16个控制点影响，这些点位于一个4×4的网格（grid）中
- Trimming and control 修剪与控制
  - The control of the surface is performed as for curves
    - the point in the control grid can be moved
    - some types of surface have tension and other parameters to use without having to move grid points
- Improving resolution 提高分辨率
  - To obtain finer detail the number of patches can be increased
  - only increase the number of patches where extra detail is needed 仅仅在需要额外细节的地方增加曲面片数量
  - 可能会导致曲面出现裂缝（cracks）

Lecture 7 3D modeling

3D modeling techniques

wireframe modeling

- The oldest and simplest approach
- The model consisting of a finite set of pointes and curves 模型由有限的点和曲线组成
- Parametric representation of a space curve
  $x = x(t), y = y(t), z = z(t)$
- Implicit representation of a space curve
  $s_1(x, y, z) = 0, s_2(x, y, z) = 0$
- combined use of curves can represent 3D objects in the space

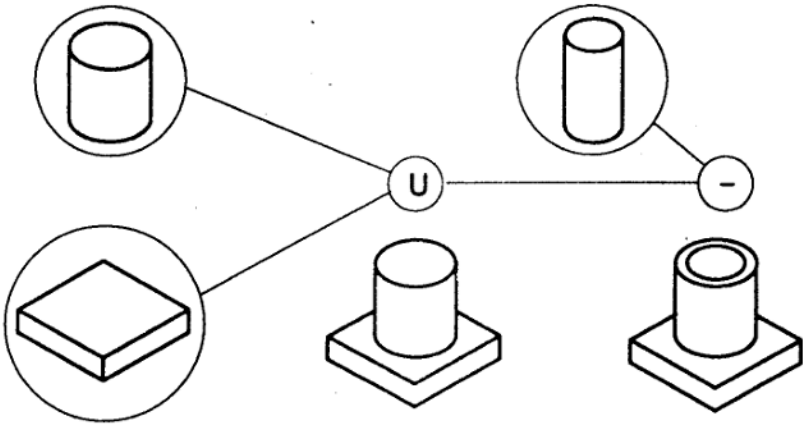
- Its disadvantages are the ambiguity of the model and the severe difficulty in validating the model 其缺点是模型存在歧义性，并且在验证模型时存在极大困难
- Furthermore, it does not provide surface and volume-related information
- wireframe Model v.s wireframe display
    - wireframe model是一种数据结构，定义物体在计算机内存中如何被定义和存储，只记录了点和线，并不知道面，同时也不能进行遮挡判断，线框显示是一种渲染模式，你可以把一个实体模型设定为线框显示模式，但他本质是包含面的

Surface modeling

- Surfaces are built from points and curves
- Surfaces can be 2D and 3D represented by a closed loop of curves with skin on it, the simplest form being a plane 曲面可以是二维和三维的 由带有蒙皮的闭合曲线环标识
- They are very important in modeling objects, and in many situations, used to represent 3D models to a large variety of satisfaction
- Despite their similar appearance, there are differences between surface and solid models
- 缺少与体积相关的信息，曲面模型通常定义了其对应物体的几何形状

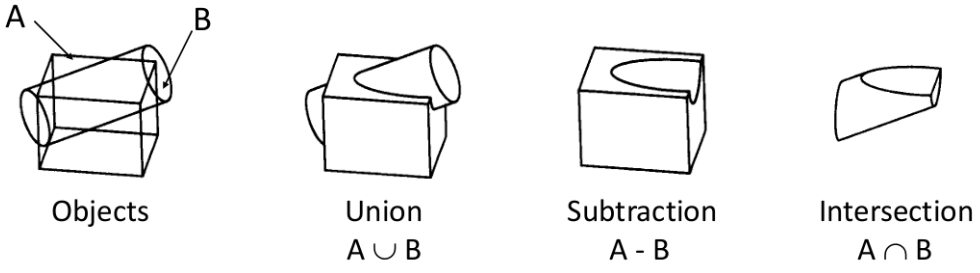
Solid modeling 实体建模

- Solid modeling represents both the geometric properties(points, curves, surfaces, volumes, and shape of shape) and physical properties(mass, centre of gravity and inertia) of solid objects
- Schemes:
    - primitive instancing 体素实体化
    - cell decomposition 单元分解
    - constructive solid geometry(CSG) 构造实体几何
        - is an ordered binary tree where the non-terminal nodes represent the operators and the terminal nodes are the primitives or transformation leaves
        - The operations may be rigid motions or regular Boolean operations 刚性运动 / 正则布尔运算
        - The primitive leaf is a primitive solid in the modeling while the transformation leaf defines the arguments of rigid motion



CSG tree

- Boolean operations include Boolean Union, Boolean Difference and Boolean Intersection
- It should be noticed that the resultant solid of a boolean operation depends not only the solids but also on their location and orientation

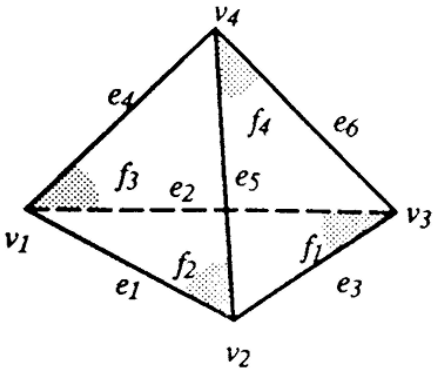


- 每个实体都有自己的局部坐标系，相对于世界坐标系指定
- 在执行布尔运算之前，可能需要对实体进行平移和旋转，以获得他们之间所需的相对位置和方向关系
- If an object can be represented by a unique set of data, the representation is said to be unique
- The representation scheme for some applications(geometric reasoning) should ideally be both unambiguous and unique 无歧义且唯一
- Solid representations are usually unambiguous but few of them are unique, and it is not feasible to make CSG representation unique

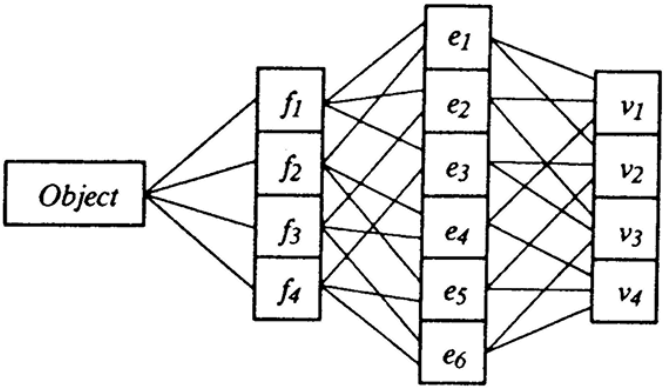
- boundary representation(B-Rep) 边界表示
  - a solid by segmenting its boundary into a finite number of bounded subsets(about the geometry and topology) B-Rep通过将其边界分割成有限数量的有界子集（关于几何与拓扑）来表示一个实体
- CSG and B-Rep are the most popular

B-Rep 边界表示

- geometry is about the shape and size of the boundary entities called points, curves and surfaces 点线面的边界实体的形状和大小
- Topology: the connectivity of the boundary entities referred as vertices, edges and faces(corresponding to points, curves and surfaces) 顶点边和面的边界实体连接性



Tetrahedron



Topology of tetrahedron

> - The same topology may represent different geometric shapes/sizes and therefore both topological and geometric data is necessary to fully and uniquely define an object 拓扑和几何数据才能完全且唯一的确定一个物体

- Types of B-Rep
  > - manifold and non-manifold 流形与非流形
  > - In a manifold model, an edge connects exactly two faces and a vertex connects at least three edges 一条边恰好连接两个面，一个顶点至少连接三条边
  > - A non-manifold model may have dangling faces, edges and vertices, and therefore represent a non-realistic / non-physical object 可能有悬空的点线面
- Euler‘s law for manifold B-Rep
  To ensure the topological validity for a solid, a manifold model must satisfy the following Euler formula

$$V - E + F - R + 2H - 2S = 0$$

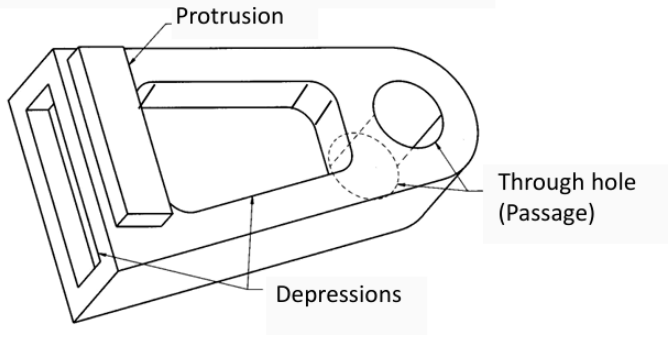
  where:
  V: vertices
  E:edges
  F:faces
  R:rings
  H:holes(passages/through-holes: genus) 通道 通孔 完全贯穿物体的隧道

S:shells(disjoint bodies), respectively 不相连的体的数量

	V	E	F	R	H	S
Basic shape	8	12	6			1
Protrusion	8	12	5	1		
Sharp corner depression	8	12	5	1		
Round corner depression	16	24	9	1		
Hole	4	6	2	2	1	
Total	44	66	27	5	1	1

$$V - E + F - R + 2H - 2S = 0$$

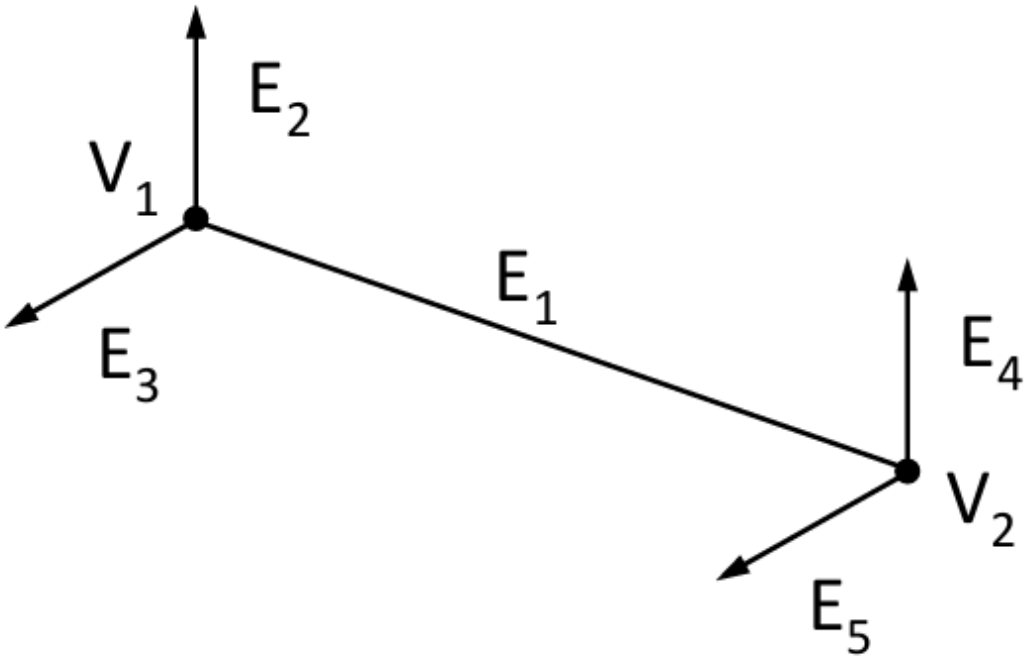
$$44 - 66 + 27 - 5 + 2x1 - 2x1 = 0$$



- 对于圆柱体，通常算成4个顶点2条边

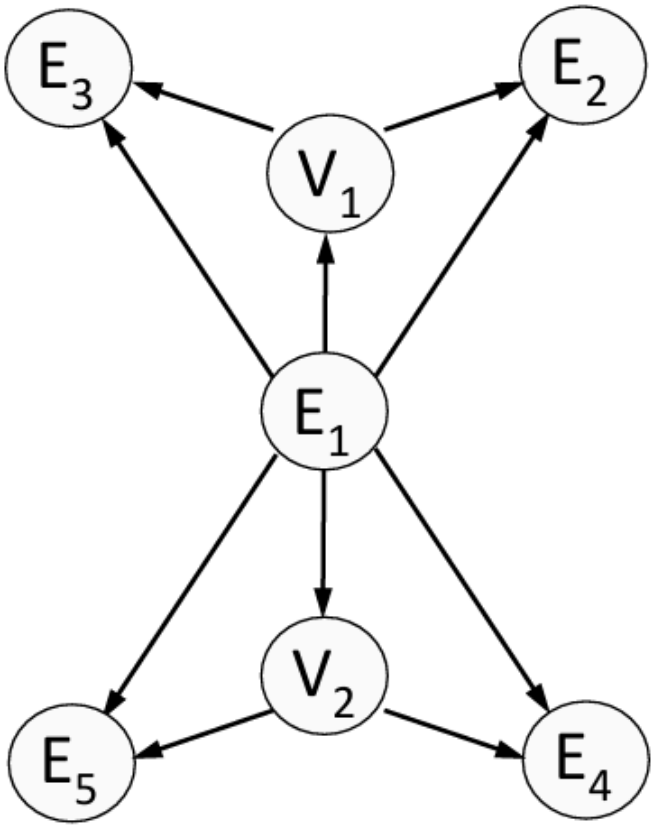
Implementation of B-Rep

- B-Rep models can be conveniently implemented on computer by representing the topology as pointers and the geometry as numerical information in the data structure for extension and manipulation using object-oriented programming(OOP) techniques 将拓扑标识为指针，将几何标识为数据结构中的数值信息



- The edge has pointers to the vertices as its ends, and to the next edges. A pointer is essentially the address in the computer's memory where something(data) is stored 该边具有其端点顶点以及指向下一条边的指针

- This structure is called `Baugmart's winged edge data structure`



Baugmart's winged edge data structure

- 之前提到的欧拉运算符其实是在修改面-边-顶点的指针结构

B-Rep Geometric Modellers

Romulus→ACIS

Lecture 8 Hierarchical Modeling

Local and World frames of reference 局部与世界坐标参考系

- The following terms are used interchangeably: Local basis | Local transformation | Local / model frame of reference
- Relative motion
  - a motion takes place relative to a local origin 相对于局部原点发生的运动
  - The local origin may be moving relative to some greater frame of reference

Linear modeling(数据的复用)

- Start with a symbol(prototype)

- Each appearance of the object in the scene is an instance 场景中物体的每次出现都是一个实例
    - we must scale, orient and position it to define the instance transformation
    - $M = T \cdot R \cdot S$
- In OpenGL
    - Set up appropriate transformations from the model frame to the world frame
    - Apply it to the `MODELVIEW` matrix before executing the code

```
glMatrixMode(GL_MODELVIEW); // M =T·R·S
glLoadIdentity();
glTranslatef();
glRotatef();
glScalef();
glutSolidCylinder(); // or other symbol
```

生成一个由很多个细长的长方形拼成一个圆柱筒

```
glBegin(GL_QUADS);
    For each A = Angles
    {
    glVertex3f(Rcos(A), Rsin(A), 0);
    glVertex3f(Rcos(A+DA), Rsin(A+DA), 0);
    glVertex3f(Rcos(A+DA), Rsin(A+DA), H);
    glVertex3f(Rcos(A), Rsin(A), H);
    }

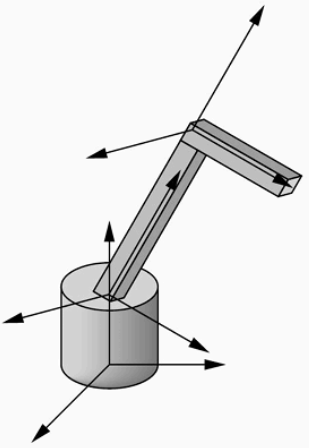
glEnd();
```

- Symbols(Primitives)
    - Cone, Sphere, GeoSphere, Teapot, Box, Tube, Cylinder, Torus, etc.
- Copy
    - Creates a completely separate clone from the original. Modifying one has no effect on the other 创建一个与原始对象完全独立的克隆 修改一个对另一个没有影响
- Instance
    - Creates a completely interchangeable clone of the original. Modifying an instanced object is the same as the modifying the original
- Array: series of clones
    - Linear
        - select object

        - define axis
        - define distance
        - define number
    - Radial 径向
        - select object
        - define axis
        - define radius
        - define number
- model stored in a table by
    - assigning a number to each symbol and
    - storing the parameters for the instance transformation
    - Contains flat information but no information on the actual structure 包括扁平化信息，当没有关于实际结构的信息

Symbol	Scale	Rotate	Translate
1	s_x, s_y, s_z	u_x, u_y, u_z	d_x, d_y, d_z
2			
3			
1			
1			
.			
.			

Linear model table



Model with constraints

Scene hierarchy

In a scene, some objects may be grouped together in some way. For example, an articulated figure may contain several rigid components connected together in a specified fashion 一些对象可能以某种方式组合在一起，例如，一个关节人物可能包含几个特定方式连接在一起的刚性组件

In each of these cases, the placement of objects is described more easily when we consider their locations relative to each other

Hierarchical modeling

Hierarchical tree

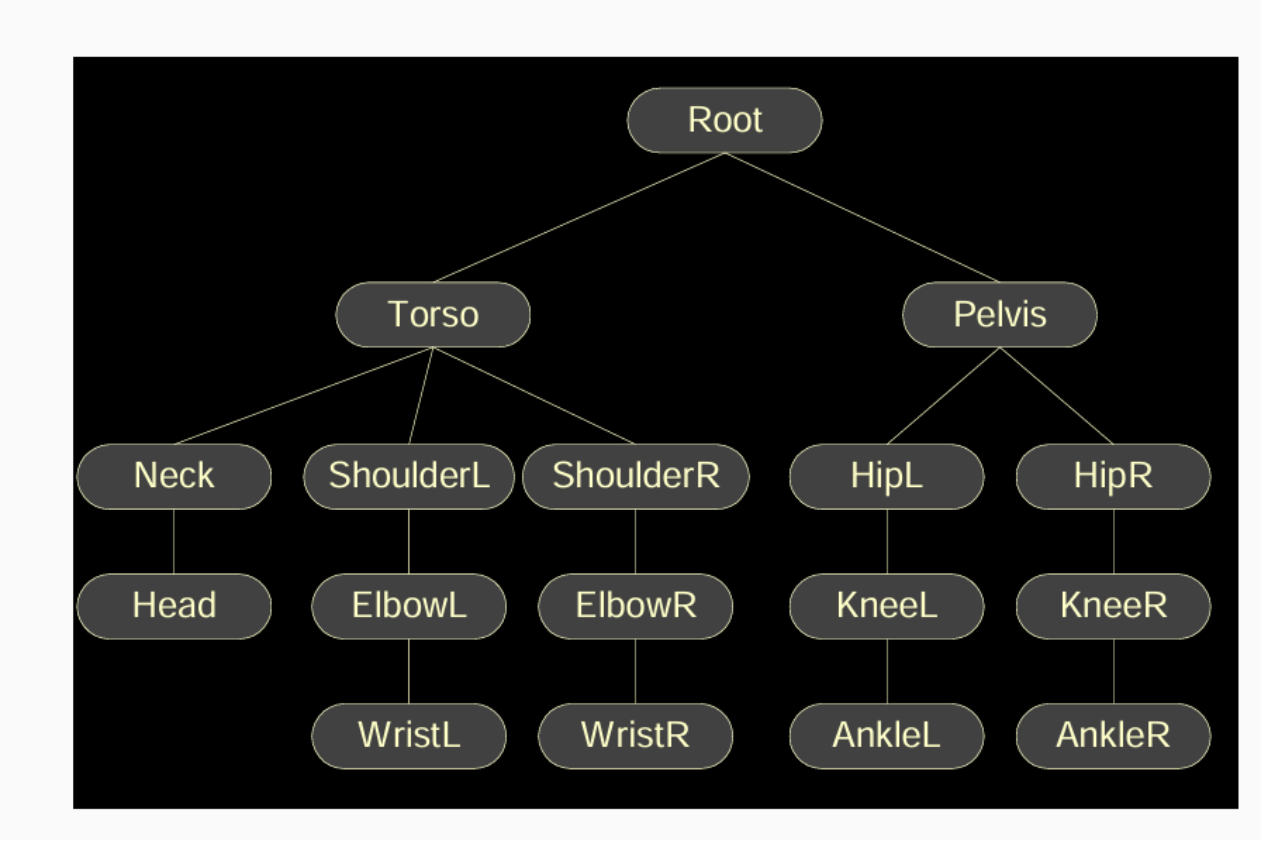
- It is very common in computer graphics to define a complex scene in some sort of

hierarchical fashion

- The individual objects are grouped into a hierarchy that is represented by a tree structure(upside down tree) 单个对象被分组到一个由数结构(倒置的树)标识的层级结构中
    - Each moving part is a single node in the tree
    - The node at the top is the root node 顶部的节点是根节点
    - Each node(expect the root) has exactly one parent node which is directly above it
    - A node may have multiple children below it
    - Nodes with the same parent are called siblings 具有相同父节点的节点叫做兄弟节点
    - Nodes at the bottom of the tree with no children are called leaf nodes 树底部没有子节点的节点称为叶节点
- Direct Acyclic Graph(DAG) stores a position of each wheel(In Car Model)
- Trees and DAGs - hierarchical methods express the relationship

Articulated model 关节模型

- 关节模型是层次模型中的一个例子，由刚性部件和连接关节组成
- 如果我们选择某个特定部件作为“root”，运动部件可以排列成树结构
- 对于一个关节模型，我们选择躯干中心附近的某个地方作为root
- 图中的每个关节都有特定的允许自由度 degrees of freedom(DOFs), 定义了模型可能的姿态范围



- Each node in the tree represents an object that has a matrix describing its location and a model describing its geometry 该对象有一个描述其位置的矩阵和一个描述其几何形状的

模型

- When a node up in the tree moves its matrix
  - It takes its children with it(in other words, rotating a character's shoulder joint will cause the elbow, wrist, and fingers to move as well)
  - so child nodes inherit transformations from their parent node 子节点继承其父节点的变换
- Each node in the tree stores a local matrix which is its transformation relative to its parent 树中的每个节点存储一个局部矩阵，它是它相对于父节点的变换
- To compute a node's world space matrix, we need to concatenate its local matrix with its parents's world matrix

$$M_{world} = M_{parent} \cdot M_{local}$$

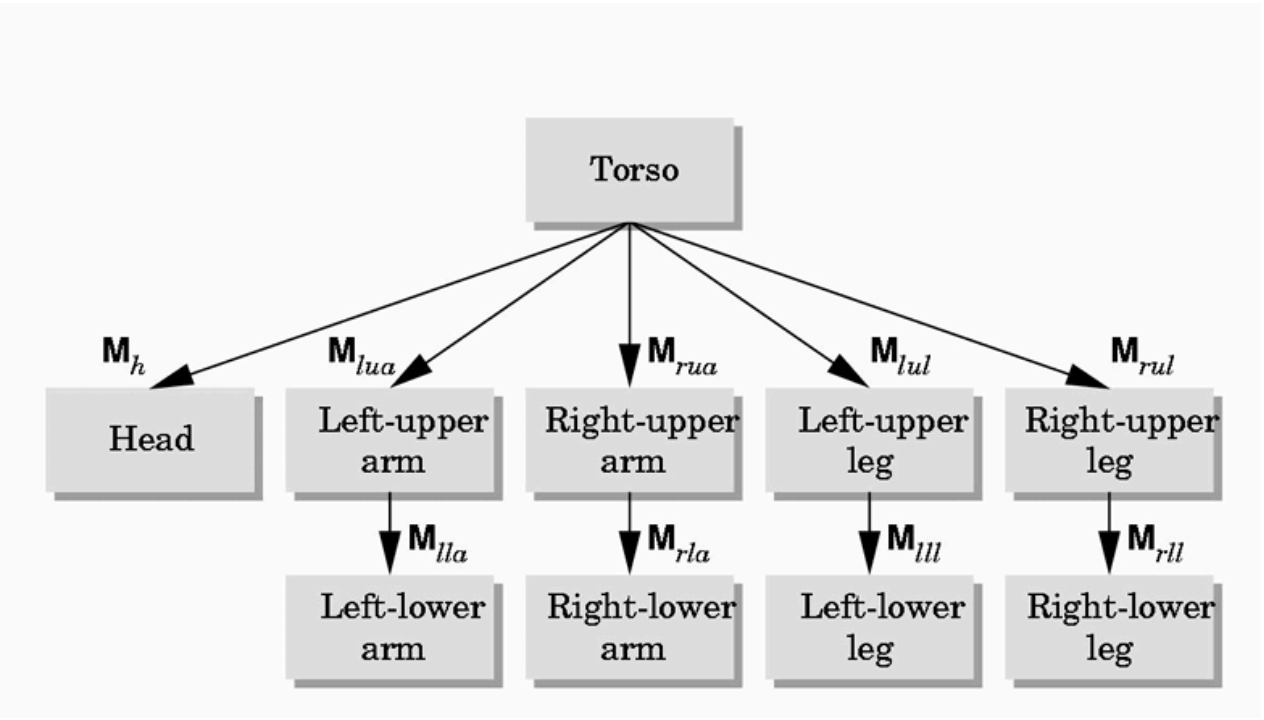
- Recursive traversal and OpenGL matrix stacks
  - To compute all of the world matrices in the scene, we can traverse the tree in a depth-first traversal
  - As each node is traversed, its world space matrix is computed 当每个节点被遍历的时，计算其世界空间矩阵
  - By the time a node is traversed, it is guaranteed that its parent's world matrix is available 当节点被遍历时，保证其父节点的世界矩阵可用
  - The GL matrix stack is set up to facilitate the rendering of hierarchical scenes
  - while traversing the tree, we can call `glPushMatrix()` when going `glPopMatrix()` when coming back up
  - Push操作：快速存档｜Pop操作：读档
- Articulated model - robot arm
  parts are connected at joints, we can specify state of model by specifying all joint angles 可以通过指定所有关节角度来指定模型状态
  - Base rotates independently
    - Single angle determines position
    - Apply $M_{b-w} = R_b$ to the base 从base到world的变换矩阵
  - Lower arm attached to the base
    - Its position depends on the rotation of the base 它的位置取决于底座的旋转
    - It must also translate relative to the base and rotate around the connecting joint 它还必须相对于底座平移并围绕连接关节旋转
    - Translate the lower arm relative to the base: T_{la}
    - Rotate the lower arm around the joint:R_{la} Apply $M_{la-w} = R_b \cdot T_{la} \cdot R_{la}$
  - Upper arm attached to lower arm
    - Its position depends on both the base and lower arm

    - It must translate relative to the lower arm and rotate around the joint connecting to the lower arm 它必须相对于下臂平移并围绕连接到下臂的关节旋转
    - Rotate the upper arm around the joint: R_{ua}, apply $M_{ua-w} = R_b \cdot T_{la} \cdot R_{la} \cdot T_{ua} \cdot R_{ua}$ to the upper arm
- Each of the 3 parts has 1 degree of freedom - described by a joint angle between them 3个部件中的每一个都有1个自由度 - 由它们之间的关节角度描述

```
void display(){
    glRotatef(theta,0.0,1.0,0.0);
    base(); // 绕Y轴旋转theta度
    glTranslatef(0.0,h1,0.0);
    glRotatef(phi,0.0,0.0,1.0)
    lower_arm();// 沿着y轴向上移动h1的距离（基座高度）然后绕着z轴旋转phi度
    glTranslatef(0.0,h2,0.0);
    glRotatef(psi,0.0,0.0,1.0);
    upper_arm(); // 沿着当前y轴向上移动h2 然后绕着z轴旋转psi度
}
```

- 模型各部分之间的关系，外观可以轻松改变而不改变关系
- If information is stored in the nodes(not in the edges), each node must store at least:
    - A pointer to a function that draws the object represented by the node
    - A matrix that positions, orients and scales the object of the node relative to the node's parent 一个矩阵，用于相对于节点的父节点（包括其子节点）定位，定向和缩放该节点的对象
    - A pointer to its children 指向其子节点的指针

- A humanoid model



  - we can build a simple implementation using quadrics: ellipsoids and cylinders
  - Access parts through functions such as
    - torso()
    - left_upper_arm()
  - Matrices describe the position of a node with respect to its parent 矩阵描述节点相对于其父节点的位置
  - Display of the tree can be thought of as a graph traversal
    - visit each node once
    - Execute the display function at each node that describes the part associated with the node, applying the correct transformation matrix for position and orientation 在每个节点处描述与该节点关联部件的显示函数，应用正确的变换矩阵进行定位和定向
  - Stack-based traversal
    - Set model-view matrix M to M_t and draw the torso
    - Set model-view matrix M to $M_t \cdot M_h$ and draw the head
    - For the left-upper arm, we need $M_t \cdot M_{lua}$ and so on
    - Rather than re-computing $M_t \cdot M_{lua}$ from scratch or using an inverse matrix, we can use the matrix stack to store $M_t \cdot M_{lua}$ and other matrices as we traverse the tree
    - ```
      void figure() {
        torso();
        glPushMatrix();
        glTranslatef();
        glRotate3();
      ```
      ```

```
    head();
    glPopMatrix();

    glPushMatrix();
    glTranslatef();
    glRotate3();
    left_upper_leg();

    glTranslatef();
    glRotate3();
    left_lower_leg();
    glPopMatrix();
    ...;
```

- 画躯干→ 存档Push（保存躯干位置）→ 变换并画头 → 读档（回到躯干）→ 再存档Pop（再次保存躯干的位置）→ 变换并画大腿
  这就导致代码变得很冗长，如果要加一个手指就得改代码结构很难维护

- Tree data structure
  - 试图把变换矩阵和层级关系存起来
  - ```
    typedef struct treenode {
      // 1. 变换矩阵：存的是“我相对于爸爸”的位置/旋转
      GLfloat m[16];
      // 2. 绘制函数：我是什么形状？（存函数指针，比如指向 torso()）
      void (*f)();
      // 3. 兄弟指针：指向我的下一个兄弟
      struct treenode *sibling;
      // 4. 孩子指针：指向我的第一个孩子
      struct treenode *child;
    } treenode;
    ```
    - 首先定义一个通用的节点来代表机器人身上的任何一个部位（头，手，脚）用左孩子-右兄弟表示法把复杂的树变成了链表结构
  - 然后初始化阶段：在这个阶段，我们不再直接画图，而是计算并存储数据
    - Torso -根节点
      - 先调用 `glRotate` 算好旋转
      - 用 `glGetFloatv` 把算好的矩阵偷出来存进 `torso_node.m`
      - 设置 `child` 指向 `head_node`（头是躯干的孩子）
      - 设置 `sibling` 为 NULL（躯干没有兄弟，它是老大）
    - 左上臂 (Left Upper Arm) - 子节点
    ```

        - 先 Translate (移到肩膀) 再 Rotate (转动手臂)
        - 把矩阵存进 `lua_node.m`。
        - 设置 sibling 指向 `rua_node`（左臂的兄弟是右臂）
        - 设置 child 指向 `lla_node`（左上臂的孩子是左小臂）
    - 最后是整个系统的引擎，通用遍历函数 Traverse
    - ```
      void traverse(treenode* root) {
           if(root == NULL) return; // 递归终止

            // 存档：保存父亲的状态
            glPushMatrix();
            // 变换：应用当前节点的矩阵（比如移动到左手位置）
            glMultMatrixf(root->m);
            // 绘制：画出当前节点（画出手臂形状）
            root->f();

            if(root->child != NULL)
             // 递归孩子：继续去画手指头（变换会累积）
                 traverse(root->child);
             // 读档：恢复到父亲的状态（抹除当前节点的变换）
             glPopMatrix();

             if(root->sibling != NULL)
             // 递归兄弟：去画右手（用的是父亲的坐标系）
                  traverse(root->sibling);
        }
      ```
        - 对孩子: 继承变换
        - 对兄弟: 不继承变换

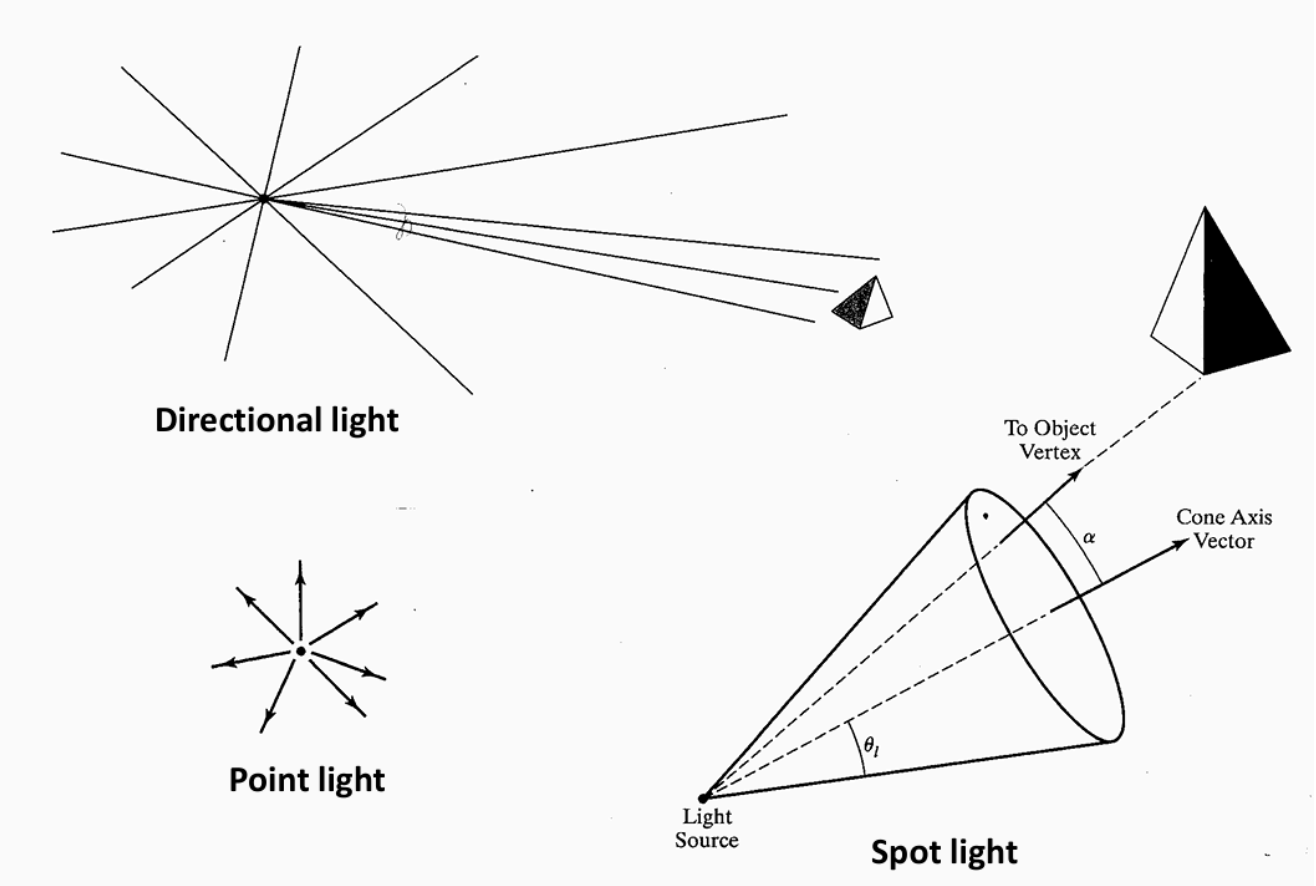
Lecture 9 Lighting and Materials

Background

- Realistic displays of a scene are obtained by generating perspective projections of objects and applying natural lighting effects to the visible surfaces 通过生成物体的透视投影并对可见表面应用自然光照效果，可以获得场景的真实感显示
- Photorealism in computer graphics also involves accurate representation of surface material properties and good physical description of the lighting effects in the scene(such as light reflection, transparency, surface texture and shadows) 图片真实感设计表面材质属性的准确表示以及对场景中光照效果的良好物理描述（光反射，透明度，表面纹理和阴影）
      ```

Light sources

光源可以用很多属性来定义，如位置，方向，颜色，形状，反射率reflectivity，可以为RGB颜色分量的每一个分配一个该颜色分量的数量(amount)和强度(intensity)



Point light

- emits radiant energy with a single color specified with the 3 RGB components 以单个颜色发出辐射能量 该颜色由三个RGB分量指定
- Large sources can be treated as point emitters if they are not too close to the scene
- The light rays are generated along radially diverging paths from the light source position 光线沿着从光源位置径向发散的路径生成

Directional light

- This is called an infinitely distant / directional light 无限远/平行光
- A large light source, which is very far away from the scene, can also be approximated as a pointer emitter, but there is little variation in its directional effects 方向性效果变化很小

Spot lights

- A local light source can easily modified to produce a spotlight beam of light 一个局部光源很容易的被修改以产生聚光灯束
- If an object is outside the directional limits of the light source, it is excluded for illumination by that light source 如果一个对象在光源的方向限制意外，则该光源将照亮该

对象

- A spot light source can be set up by assigning a vector direction and an angular limit θ_i measured from the vector direction along with the position and color 聚光灯光源可以指定一个向量方向和一个从向量方向测量的角度限制来设置，同时还需要指定位置和颜色

Surface lighting effects 表面光照效果

- Although a light source delivers a single distribution of frequencies, the ambient, diffuse and specular components might be different 尽管光源提供单一频率分布，但环境光 漫反射 高光反射分量可能不同

Ambient light 环境光

- is a general background non-directed illumination. Each object is displayed using an intensity intrinsic to it, i.e. a world of non-reflective, self-luminous object 这种效果是一种通用的，无方向的背景照明，每个对象都使用其固有的强度显示，即一个非反射的自发光的世界
- Consequently each object appears a monochromatic silhouette, unless its constituent parts are given different shades when the object is created 因此每个对象都显示为单色轮廓，除非其组成部分在创建对象时被赋予不同的色调
- The ***ambient component*** is the lighting effect produced by the reflected light from various surfaces in the scene 环境光分量是由场景中各种表面反射光产生的光照效果
- When ambient light strikes a surface, it is scattered equally in all directions
- The light has been scattered so much by the environment that it is impossible to determine its direction - it seems to come from all directions
- 房间中的背光有很大的环境光分量，因为到达眼睛的大部分光线首先经过了许多表面的反射，户外的聚光灯环境光分量很少，大部分光线沿相同方向传播，并且由于在户外，很少有光线反弹到其他物体后到达眼睛

Diffuse reflectance 漫反射光

- Diffuse light comes from one direction, so it is brighter if it comes squarely down on a surface than if it barely glances off the surface 漫反射光来自一个方向，所以如果它垂直照射表面，会比它擦过表面时更亮
- Once it hits a surface, however, it is scattered equally in all directions, so it appears equally bright, no matter where the eye is located 一旦它击中表面，它会在所有方向上均匀散射，所以无论眼睛位于何处，它看起来一样亮
- Any light coming from a particular position or direction probably has a diffuse component
- Rough or grainy surfaces tend to scatter the reflected light in all directions(called diffuse reflection) 粗糙或有纹理的表面倾向于在所有方向上散射反射光（漫反射）
- 每个对象都被认为是呈现暗淡，无光泽的表面，并且所有对象都由一个点光源照亮，该光源向所有方向上均匀发出

- The factors which affect the illumination are the point light source intensity, the material's diffuse reflection of the light影响照明的因素包括点光源的强度，材质的漫反射系数以及光线的入射角

Specular reflectance 高光反射光

- Specular light comes from a particular direction, and it tends to bounce off the surface in a preferred direction 高光来自于特定方向，并且他倾向于在首选方向上从表面反弹
- 一个高度准直的激光束从高质量镜子上反弹产生几乎100%的高光反射，高亮的金属或塑料具有很高的高光分量，而粉笔或地毯则几乎没有，这种效果可以在任何闪亮的表面上看到，从表面反射光往往具有与光源相投的颜色
- Specularity can be considered as shininess 光泽度
- The reflectance tends to fall off rapidly as the viewpoint direction veers away from the direction of reflection (which is equal to the direction to the light source mirrored about the surface normal); for perfect reflectors (i.e. perfect mirrors), specular light appears only in the direction of reflection.当视点方向偏离反射方向时，反射率倾向于迅速下降，对于完美反射体，高光只出现在反射方向
- The factors that affect the amount of light reflected are the material’s specular-reflection coefficient, the angle of viewing, relative to the direction of reflection, and the light source intensity.材质的高光反射系数 相对于入射方向的视角以及光源强度

Attenuation 衰减

- For point/positional light sources, we consider the attenuation of light received due to the distance from the source
- Attenuation is disabled for directional lights as they are infinitely far away, and it does not make sense to attenuate their intensity over distance 因为他们是无限远的，所以对其强度进行距离衰减没有意义
- Although for an ideal source the attenuation is inversely proportional to the square of the distance d, we can gain more flexibility by using the following distance-attenuation model

$$f(d) = \frac{1}{k_c + k_1 d + k_q d^2}$$

Lighting model

- A lighting model(also called illumination or shading model) is used to calculate the color of an illuminated position on the surface of an object. 用于计算表面被照亮位置的颜色
- The lighting model computes the lighting effects for a surface using various optical properties, which have been assigned to the surface, such as the degree of transparency, color reflectance coefficients and texture parameters. 光照模型使用已分配给表面的各种光学属性（透明度，透明反射系数，纹理参数）来计算表面的光照效果

- The lighting model can be applied to every projection position, or the surface rendering can be accomplished by interpolating colors on the surface using a small set of lighting-model calculations.可以应用于每个投影位置，或者可以通过使用少量光照模型计算来插值表面上的颜色来完成表面渲染
- A surface rendering method uses the color calculation from a lighting model to determine the pixel colors for all projected positions in a scene. 表面渲染方法是用来制光照模型的颜色计算来确定场景中所有投影位置的像素颜色

Phong model

- A simple model that can be computed rapidly
- 3 components considered
    - Diffuse 漫反射
    - Specular 高光反射
    - Ambient 环境光
- 4 vectors used
    - To light source l
    - To viewer v
    - Face normal n
    - Perfect reflector r （完美反射体）
- For each point light source, there are 9 coefficients
  $I_{dr}, I_{dg}, I_{db}, I_{sr}, I_{sg}, I_{sb}, I_{ar}, I_{ag}, I_{ab}$
- Material properties match light source properties
    - 9 absorption coefficients 吸收系数
      $k_{dr}, k_{dg}, k_{db}, k_{sr}, k_{sg}, k_{sb}, k_{ar}, k_{ag}, k_{ab}$
    - Shininess coefficient α 光泽度系数
- For each light source and each color component, the Phong model can be written(without the distance terms) as

$$I = k_d I_d \cdot l \cdot n + k_s I_s (v \cdot r)^{\alpha} + k_a I_a$$

- For each color component, we add contributes from all sources

Shading

Polygonal shading

- Phong Model 是针对每一个顶点进行的 而不是像素，点那个顶点颜色被计算出来后把属性发给顶点着色器，或者让顶点着色器直接计算颜色
- shade: 该店在光照影响下呈现出的最终外观（颜色加亮度）
- In vertex shading, shading calculations are done for each vertex

> - Vertex colors become vertex shades and can be sent to the vertex shader as a vertex attribute 顶点颜色变为顶点阴暗值，可以作为顶点属性发送给顶点着色器
> - Alternately, we can send the parameters to the vertex shader and have it compute the shade 可以将参数发送到顶点着色器，并由他计算阴暗值

- （现代图形学默认处理方式）By default, vertex shades are interpolated across an object if passes to the fragment shader as a varying variable(smooth shading), we can also use uniform variables to shade with a single shade(flat shading) 默认情况下，如果顶点颜色计算好了计算好了，但这些数据传递给片元着色器时，系统会自动在多边形内部进行插值（如果一个三角形三个顶点时红绿蓝，中间的像素会自动呈现出红绿蓝平滑过渡的渐变色） 同时我们也可以使用统一变量（uniform variables）或单一的颜色属性来为整个多边形着色

Flat(constant) shading 平面（恒定）着色

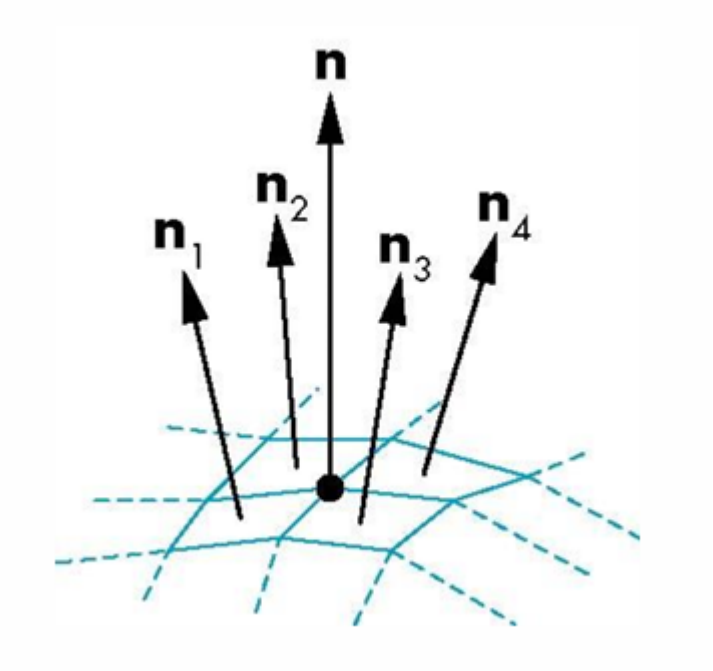
- 如果三个向量 (l, v, n) 是常量，那么着色系数只需要对每个多边形执行一次，并且多边形上每个点都被分配成相同的shade，实际上他理想化地把多边形上的每一个点的法向量想成相同的，这样就会导致每个多边形的颜色是一样的
- 多边形网格通常被设计为模拟光滑表面，而平面着色总是令人失望的，因为我们深知能看到相邻多边形之间阴暗值的微小差异

Smooth(interpolative) shading 平滑（线性）着色

- The rasteriser interpolates colors assigned to vertices across a polygon 光栅化器在多边形上插值分配给顶点的颜色

- Suppose that the lighting calculation is made at each vertex using the material properties and vectors n, v, and l computed each vertex. Thus, each vertex will have its own color that the rasteriser can use to interpolates a shade for each fragment 每个顶点都有自己的颜色，光栅化器可以使用这些颜色为片元插值出一个敏感字
- Note that if the light source is distant, and either the viewer is distant or there are no specular reflections, then smooth(or interpolative) shading shades a polygon in a constant color 如果光源是远距离的，并且观察者也是远距离的或者没有高光反射，那么平滑（或插值）着色会用恒定颜色对多边形进行着色

Gouraud Eshading



- The vertex normals are determined(average of the normals around a mesh vertex: $n = \frac{n_1+n_2+n_3+n_4}{|n_1+n_2+n_3+n_4|}$ for a polygon and used to calculate the pixel intensities at each vertex, using whatever lighting model 确定顶点法线，对于任何光照模型计算每个顶点的像素强度
- Then the intensities of all points on the edges of the polygon are calculated by a process of weighted averaging of the vertex values 然后通过顶点的加权平均（线性插值）过程计算多边形边上所有点的强度
- The vertex intensities are linearly interpolated over the surface of the polygon 顶点强度在多边形表面上进行线性插值

RGB values for lights and materials

- The color components specified for lights mean something different from those for materials 为光源指定的颜色分量含义与为材质指定的不同
- For a light, the numbers correspond to a percentage of full intensity for each color. If the R, G, B values for a light color are all 1.0, the light is the brightest white. 该光源是最亮的白色

- If the values are 0.5, the color is still white, but only at half intensity, so it appears grey. If R=G=1 and B=0(full red and green with no blue), the light appears yellow 光源看起来是黄色
- For materials, the numbers corresponding to the reflected proportions of those colors. So if R=1, G=0.5 and B=0 for a material, the material reflects all the incoming red light, half the incoming green and none of the incoming blue light 对于材质，数字对应这些颜色的反射比率，因此一个材质的RGB表示对这些光的反射
- In other words, if a light has components (R_L, G_L, B_L), and a material has corresponding components (R_M, G_M, B_M); then, ignoring all other reflectivity effects, the light that arrives at the eye is given by $(R_L R_M, G_L G_M, B_L B_M)$
- Similarly, if 2 lights (R_1, G_1, B_1) and (R_2, G_2, B_2) are sent to the eye, these components are added up, giving $(R_1 + R_2, G_1 + G_2, B_1 + B_2)$
- If any sum≥ 1, the component is clamped to 1

Material properties

- A lighting model may approximate a material color depending on the percentages of incoming red, green, and blue light reflected 根据入射红绿蓝光的反射百分比，光照模型可以近似材质颜色
  > - 一个完美的红球反射所有入射的红色光，吸收所有照射到它上面的绿色和蓝色光
- If the ball is lit in white light, all the red is reflected, and a red ball is seen. If the ball is viewed in pure red light, it also appears to be red 在白光下 所有的红色光北反射 所以还是红球，在红光下还是红球，但是如果在纯绿光下应该是黑球
- Like lights, there are ambient, diffuse and specular reflections for materials
- The ambient reflectance of a material is combined with the ambient component of each incoming light source, the diffuse reflectance with the light's diffuse component, and similarly for the specular reflectance component 材质的环境光反射率与每个入射光源的环境光分量相结合，漫反射率与光源的反射率分量相结合...
- 环境光和漫反射率定义了材质的颜色，并且通常相似，即使不完全相同，高光反射率通常是白色或者灰色
- Diffuse reflectance plays the most important role in determining what we perceive the color of an object 漫反射率在决定我们感知物体的颜色起着最重要的颜色
- Ambient reflectance affects the overall colour of the object. Because ambient reflectance is the brightest where an object is directly illuminated, it is the most noticeable where an object receives no direct illumination. The total ambient reflectance for an object is affected by the global ambient light and ambient light from individual light sources. 环境光反射率影响物体的整体颜色 因为环境光反射率在物体被直接照亮的地方最亮，随意在物体没有受到直接照明的地方最为明显
- Diffuse and ambient reflectances are not affected by the position of the viewpoint. 漫反射和环境光反射率不受视点位置影响
- Specular reflection from an object produces highlights

- Unlike ambient and diffuse, the mount of specular reflection seen by a viewer depends on the location of the viewpoint

OpenGL

Light sources and color

- Light source have a number of properties, such as color, position and direction
- The command used to specify all properties of lights is `glLight*()`
- It takes 3 arguments: the identity of the light, the property and the desired value for that property
- `void glLight{if}[v](GLenum light, GLenum pname, TYPEparam` creates the light specified by light
- The characteristic of the light being set is defined by `pname` 命名参数

Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient RGBA intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	diffuse RGBA intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	specular RGBA intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(*x, y, z, w*) position of light
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	(*x, y, z*) direction of spotlight
GL_SPOT_EXPONENT	0.0	spotlight exponent
GL_SPOT_CUTOFF	180.0	spotlight cut-off angle
GL_CONSTANT_ATTENUATION	1.0	constant attenuation factor
GL_LINEAR_ATTENUATION	0.0	linear attenuation factor
GL_QUADRATIC_ATTENUATION	0.0	quadratic attenuation factor

- The `TYPEparam` argument indicates the values to which the `pname` characteristic is set 如果使用向量版本，它是一组值的指针，如果使用非向量版本，则是值本身，非向量版本只能设置单值光源特性

Light position and direction

- A light source can be treated as through it is locates infinitely far away from the scene or close to the scene 光源可以被视为位于场景无限远处或无限靠近场景
- The first is referred to as a directional light source; the effect of an infinite location is that the rays of lights can be considered parallel 平行光源
- The second is called a positional/point light source, since its exact position within the scene determines its effect on the scene and, specifically, the direction from which the

light rays come. A desk lamp is an example of a positional light source. 位置/点光源 他在场景中确切位置决定了其对场景的影响

- The light specified below is directional (where w = 0).
  `GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0}`
  `glLightfv(GL_LIGHT0, GL_POSITION, light_position)`
- If the w value is 0.0, the corresponding light source is a directional one, and the (x, y, z) values describe its direction, This direction is transformed by the model-view matrix
- If the w value is nonzero, the light is position, and the (x, y, z) values specify the location of the light in homogenous object co-ordinates
- This location is transformed by model-view matrix and stored in eye co-ordinates

Spotlight

- Note that no light is emitted beyond the edges of the cone.
- By default, the spotlight feature is disabled because the `GL_SPOT_CUTOFF` parameter is 180.0. This value means that light is emitted in all directions (the angle at the cone's apex is 360 degrees, so it is not a cone at all).
- The value for `GL_SPOT_CUTOFF` is restricted to being within the range [0.0,90.0] (unless it has the special value 180.0).
- The following line sets the cut-off parameter to 45 degrees: `glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);`
- We can apply distance attenuation described earlier. 我们可以应用前面描述的距离衰减
- We can also set the `GL_SPOT_EXPONENT` parameter, to control how concentrated the light is.控制光的集中程度
- The light intensity is the highest in the centre of the cone. It is attenuated towards the edges of the cone by the cosine of the angle between the direction of the light and the direction from the light to the vertex lit exponentially. Thus, a higher spot exponent results in a more focused light source. 光的强度在锥体中心最高
- As directional and point lights, spotlight direction is transformed by the model-view matrix just as though it were a normal vector 和平行光与点光源一样 由model-view 矩阵变换

Multiple lights

- As aforementioned, we can have up to 8 lights in our scene (possibly more, depending on your OpenGL implementation). 场景中最多有8个光源
- Since OpenGL needs to perform calculations to determine how much light each vertex receives from each light source, increasing the number of lights adversely affects performance

Global Ambient Light

- Each light source can contribute ambient light to a scene. There can be other ambient light that is not from any particular source.
- The `GL_LIGHT_MODEL_AMBIENT` parameter is used to specify the RGBA intensity of such global ambient light
  ```
  GLfloat lmodel_ambient[] = {0.2, 0.2, 0.2,1.0};
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
  ```
- In the example, the values used for `lmodel_ambient` are the default values for `GL_LIGHT_MODEL_AMBIENT`. Since these numbers yield a small amount of white ambient light, even if we do not add a specific light source to our scene, we can still see the objects in the scene

Local or infinite Viewpoint

- The location of the viewpoint affects the calculations for highlights produced by specular reflectance(as shown in the Phong model) 视点位置影响高光反射产生的高光的计算
- A local viewpoint tends to yield more realistic results, but overall performance is decreased because the direction has to be calculated for each vertex. 局部视点倾向于产生更真实的结果
- With an infinite viewpoint, the direction between it and any vertex in the scene remains constant, thus requiring less computational power. 对于无限远视点 他与场景中的任何顶点的方向保持恒定
- By default, an infinite viewpoint is assumed. Here is how to change to a local viewpoint `glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);`
- This call places the viewpoint at (0, 0, 0). To switch back to an infinite viewpoint, pass in `GL_FALSE` as the argument.

Two-sided Lighting

- We usually set up lighting conditions for the front-facing polygons, and the back-facing polygons typically are not correctly illuminated. 通常为正面多边形设置光照条件，而背面多边形通常无法正确照亮
- If the object is a sphere, only the front faces are ever seen because they are the ones on the outside of the sphere. So, in this case, it does not matter what the back-facing polygons look like.
- If the sphere is cut away so that its inside surface would be visible, however, we may want to have the inside surface fully lit according to the lighting conditions defined; we may also want to supply a different material description for the back faces.

Defining material properties

- Most of the material properties are conceptually similar to those used to create light sources
- The mechanism for setting them is similar, except that the

- command used is called `glMaterial*()`,
  e.g. `void glMaterial{if}[v](GLenum face, GLenum pname,TYPEparam)`

Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	ambient colour of material
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	diffuse colour of material
GL_AMBIENT_AND_DIFFUSE		ambient and diffuse colour of material
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	specular colour of material
GL_SHININESS	0.0	specular exponent
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	emissive colour of material
GL_COLOR_INDEXES	(0,1,1)	ambient, diffuse and specular colour indices

- The particular material property being set is identified by `pname` and its desired values are given by `TYPEparam`, which is either a pointer to a group of values (if the vector version is used) or the actual value (if the non-vector version is used, only for setting `GL_SHININESS`)
- `GL_EMISSION` 指定RGBA颜色 可以使对象看出来发出该颜色的光 模拟场景中的灯和其他发光物体
- face参数可以是 `GL_FRONT`, `GL_BACK` OR `GL_FRONT_BACK` 以指示材质应用于对象的那个面

Lecture 10 Texture Mapping

Concepts of texture mapping

A common method for adding detail to an object is to map patterns onto the geometric description of the object. 向对象添加细节的常用方法是将团映射到对象的几何描述上，将对象细节融入场景的方法称为纹理映射

Tapes of Texture mapping

- Texture mapping 使用图像填充多边形内部
- Environment mapping 使用环境图片作为纹理贴图 允许模拟高光表面
- Bump mapping 模拟在渲染过程中改变发现向量

Specifying the texture

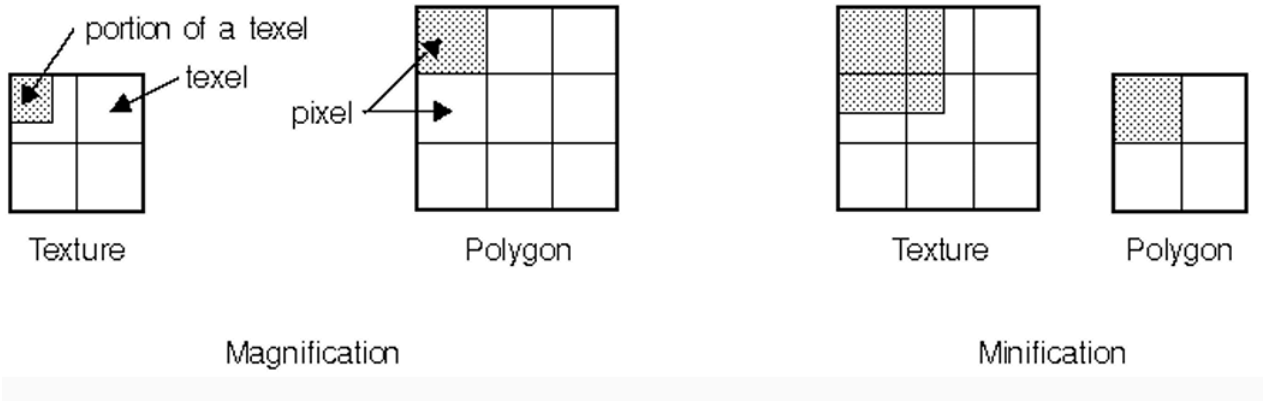
2 common types of texture

- Image: a 2D image is used as the texture
- Procedural: a procedure or program generates the texture

- 图形包中的纹理函数通常允许将图案中每个位置的颜色分量数量指定为选项
- 纹理图案中的每个颜色定义可以包含为四个RGBA分量，3个RGB分量，用于蓝色色调的单个强度值，颜色表中的索引或单个亮度值
- 纹理数组中的各个值通常称为纹素（texels）

Magnification and minification

- Depending on the texture size, the distortion and the size of the screen image, a texel may be mapped to more than one pixel (called magnification), and a pixel may be covered by multiple texels (called minification) 根据纹理大小，失真和屏幕图像大小，一个纹素可能映射到多个像素（放大），或者一个像素被多个纹素覆盖（缩小）



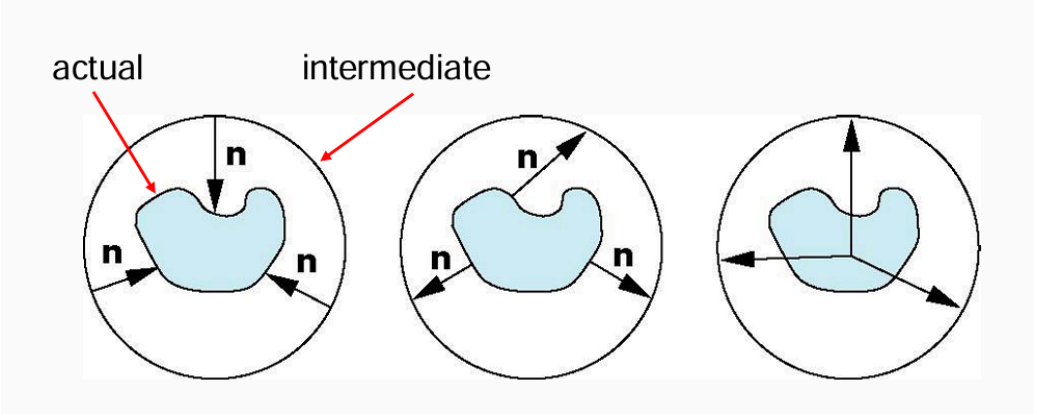
- 由于纹理由离散的纹素组成，必须执行过滤操作以将纹素映射到像素
- Texture mapping techniques are implemented at the end of the rendering pipeline 纹理映射技术发生在渲染管线的末端
- It is high efficiency because a few polygons make it past the clipper
- Co-ordinate systems for texture mapping
    - Parametric co-ordinates
        - maybe used to model curves and surfaces
    - Texture co-ordinates
        - is used to identify points in the image to be mapped
    - Object or world co-ordinates
        - conceptually, where the mapping taking place 概念上映射发生的地方
    - Window co-ordinate
        - where the final image is finally produced 最终图像生成的地方
    - Considering mapping from texture co-ordinates to point on a surface

- Apparently 3 functions are needed
  - $x = x(s,t)$
  - $y = y(s,t)$
  - $z = z(s,t)$
- But we really want to go the other way
- Backward mapping: 给定一个像素 我们想知道对应对象上的哪个点 给定对象上的恶点 我们想知道他对应的纹理中的哪个点 我们更想找到这样的映射 即使很困难
  - $s = s(x,y,z)$
  - $t = t(x,y,z)$

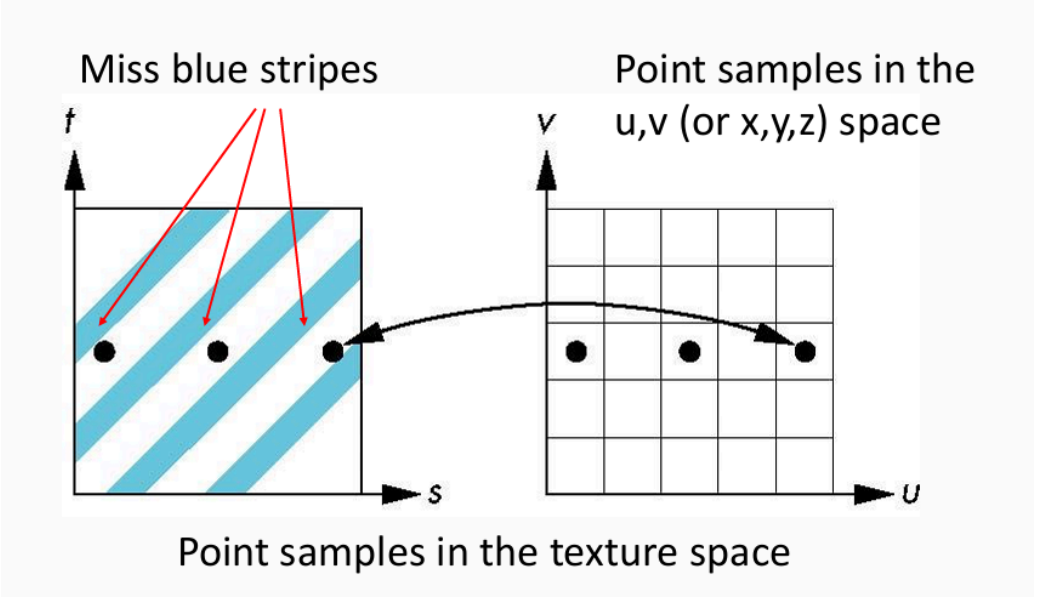
Two-step mapping

- One solution to the mapping problem is to first map the texture to a simple intermediate surface 先映射到简单的中间平面
- e.g. mapping into a Cylinder
  - A parametric cylinder
    - $x = r * \cos(2pi * u)$
    - $y = r * \sin(2pi * u)$
    - $z = h * v$
    - maps a rectangle in the (u,v) space to the cylinder of radius r and height h in the world co-ordinates $s = u, t = v$ maps from the texture space
  - mapping into a parametric sphere
    - $x = r * \cos(2pi * u)$
    - $y = r * \sin(2pi * u) * \cos(2pi * v)$
    - $z = r * \sin(2pi * u) * \sin(2pi * v)$
    - in a similar manner to the cylinder but have to decide where to put the distortion
  - Box mapping
    - Easy to use with simple orthographic projection 正交投影
    - Also used in environmental maps
- Second mapping
  - Mapping from an intermediate object to an actual object 从中心对象到实际对象的映射
    - Normals from the intermediate to actual objects 从中间对象到实际对象的法线
    - Normals from the actual to intermediate objects 从实际对象到中间对象的法线

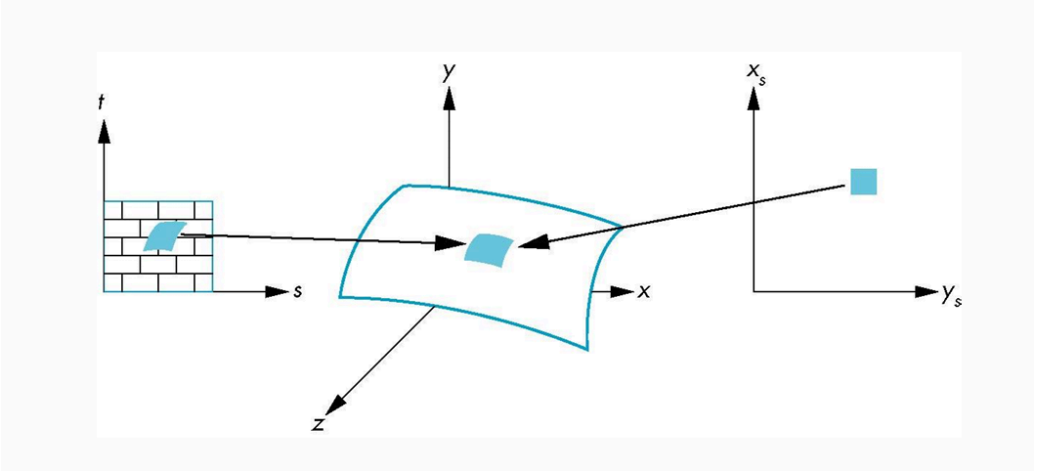
- Vectors from the center of the intermediate object 从中间对象的中心出发的向量



- Aliasing
  - Point sampling of the texture can lead to aliasing errors



  - Area averaging: a better but slower option



OpenGL steps in texture mapping

3 main steps to applying a texture

- Specify the texture

    - Read or generate image
    - Assign to texture
    - Enable texturing
- Assign texture coordinates to vertices
    - Proper mapping function is left to application
- Specify texture parameters
    - Mode
    - Filtering
    - Wrapping

OpenGL functions

- `glTexImage2D()`: Defining image as texture (type, level and color) 将图像定义为纹理
- `glTexParameter*()`: Specifying filtering and wrapping 指定过滤和环绕
- `glTexEnv{fi}[v]()`: Specifying mode (modulating, blending or decal) 指定模式（调制混合和贴花）
- `glTexCoord{1234}{sifd}{v}()`: Specifying texture co-ordinates s and t while r is set to 0 and q to 1 指定纹理坐标s和t 同时r设为0，q设为1
- `glTexGen{ifd}[v]()`: Automatic generation of texture co-ordinates by OpenGL 由OpenGL自动生成纹理坐标
- `glHint()`: Defining perspective correction hint 定义透视矫正
- `glBindTexture()`: Binding a named texture to a texturing target 将命名纹理绑定到纹理目标

Texture mapping and the OpenGL pipeline

- The images and geometry flow through separate pipelines that join during fragment processing 图像和几何数据通过独立的管线流动，在片元处理阶段混合
- Complex textures do not effect geometric complexity

Specifying a texture image 指定纹理图片

- Define a texture image from an array of texels(texture elements) in CPU memory
    - `Glubyte my_texels[512][512][4]`
- Define a texture as any other pixel map

    - scanned image
    - generate by an application program
- Enable texture mapping
    - `glEnable(GL_TEXTURE_2D)`
    - OpenGL supports 1-4 dimensional texture maps

Defining image as a texture 将图像定义为纹理
`glTexImage2D(target,level,components, w,h,border,format,type, texels)`

`target`: type of texture, e.g. `GL_TEXTURE_2D`

`level`: used for mipmapping (0 for only one/top resolution – to be discussed shortly)

`components`: color elements per texel- an integer from 1 to 4 indicating which of the R, G, B and A components are selected for modulating or blending. A value of 1 selects the R component, 2 selects the R and A components, 3 selects R, G and B, and 4 selects R, B, G and A 每个纹素的颜色元素-从1-4的整数 表示选择RGBA分量中的哪些进行调制和或混合，1选择R，2选择 RA 3选择RGB 4选择RGBA

`w and h`: width and height of the image in pixels. 图像的宽度和长度（像素）

`border`: used for smoothing (discussed shortly), which is usually 0.

Both w and h must have the form $2^m + 2b$, where m is an integer and b is the value of border though they can have different values. The maximum size of a texture map depends on the implementation of OpenGL, but it must be at least 64 × 64 (or 66 × 66 with borders)

`format and type`: describe the format and data type of the texture image data. They have the same meaning with `glDrawPixels()`. In fact, texture data has the same format as the data used by `glDrawPixels()`.
The format parameter can be `GL_COLOR_INDEX`, `GL_RGB`, `GL_RGBA`, `GL_RED`, `GL_GREEN`, `GL_BLUE`, `GL_ALPHA`, `GL_LUMINANCE`, or `GL_LUMINANCE_ALPHA`
– i.e., the same formats available for `glDrawPixels()` with the exceptions of `GL_STENCIL_INDEX` and `GL_DEPTH_COMPONENT` 描述纹理图像数据的格式和数据类型，它们与 `glDrawPixels` 具有同样的含义

Filtering, wrapping and modes

OpenGL has a variety of parameters that determine how texture is applied

- Filter modes allow us to use area averaging instead of point samples. 过滤模式允许我们使用区域平均而不是点采样
- Wrapping parameters determine what happens if s and t are outside the [0,1] range. 决定

当s和t超出范围时会发生什么

- Mode/environment parameters determine how texture mapping interacts with shading.决定纹理映射如何与着色交互
- Mipmapping(discussed shortly) allows us to use textures of multiple resolutions (levels of detail). 允许我们使用多重分辨率的纹理

Controlling filtering

- 通过使用 `glTextParameter*()` 来指定放大和缩小的过滤方法
  `glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_NEAREST)`
  `glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER, GL_LINEAR)`
- 第二个参数是 `GL_TEXTURE_MAG_FILTER` 或 `GL_TEXTURE_MIN_FILTER` 用来指定放大还是缩小的方法，第三个参数指定过滤方法
- 线性过滤在边缘过滤时需要额外的纹素边框（boarder = 1）

Texture wrapping

- We can assign texture co-ordinates outside the range [0,1] and have them either clamp or repeat in the texture map. 我们可以分配超出范围的纹理部分
- With repeating textures, if we have a large plane with texture co-ordinates running from 0.0 to 10.0 in both directions, for example, we will get 100 copies of the texture tiled together on the screen. During repeating, the integer part of texture co-ordinates is ignored, and copies of the texture map tile the surface. 重复纹理 重复纹理坐标的整数部分被忽略 纹理映射的副本平铺在表面上
- For most applications of repeating, the texels at the top of the texture should match those at the bottom, and similarly for the left and right edges
- The other possibility is to clamp the texture co-ordinates: any values greater than 1.0 are set to 1.0, and any values less than 0.0 are set to 0.0 钳制纹理坐标 单个纹理副本出现在大平面上很有用
  `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,GL_CLAMP)`
  `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,GL_REPEAT)`

Texture mode

- Control how texture is applied
- Set blend color with `GL_TEXTURE_ENV_COLOR`
- `glTexEnv{if}{v}(GLenum target, GLenum pname, TYPEparam),`
    - 直接使用纹理颜色作为最终颜色 称为贴花模式（decal mode）其中纹理就像贴纸一样会自在片元顶部
    - 使用纹理来调制或缩放片元的颜色，这对于将光照效果与纹理结合很有用
    - 基于纹理值 可以将一个恒定颜色与片元颜色混合
    - target必须是 `GL_TEXTURE_ENV`
    - 如果 `pname` 是 `GL_TEXTURE_ENV_MODE`，`TYPEparam` 可以是以下的
    - `GL_TEXTURE_ENV_MODE` modes
        - `GL_DECAL`
        - `GL_REPLACE` : use only texture color 仅使用纹理颜色
        - `GL_MODULATE` : modulates with computed shade 与计算出的阴暗值调制正片叠底
        - `GL_BLEND` : blends with an environmental color 与环境颜色混合
    - 在贴花模式下使用三分量纹理时，纹理颜色替换片元颜色
    - 在其他两种模式下 使用四分量纹理时，最终颜色是纹理值和片元值的组合 如果 `pname` 是 `GL_TEXTURE_ENV_COLOR`，则 `TYPEparam` 是一个包含RGBA四分量 仅当同时制定了 `GL_BLEND` 纹理函数时，才会用到

Mipmapping - Multiple levels of detail

- 纹理对象可以像场景中的其他对象一样，从视点在不同距离处被观察，在动态场景中，随着纹理对象离视点越来越远，纹理映射必须随着投影大小而缩小
- OpenGL automatically determines which texture map to use based on the size (in pixels) of the object being mapped. 纹理图中的细节层次适合绘制在屏幕上的图像 随着对象图像变小，纹理图的大小也随之变小
- With this approach, the level of detail in the texture map is appropriate for the image that is drawn on the screen - as the image of the object gets smaller, the size of the texture map decreases.
- Mipmapping requires some extra computation to reduce interpolation errors, but, when it is not used, textures that are mapped onto smaller objects might shimmer and flash as the objects move
- Mipmapping level is declared during texture definition by calling `glTexImage2D(GL_TEXTURE_*D, level, ...)` once for each resolution of the texture map, with different values for the level, width, height and image
- 此外还应该选择适合的过滤方式

Generating texture co-ordinates

- Texture co-ordinates are usually referred to as the s, t, r and q co-ordinates to distinguish them from object co-ordinates (x, y, z and w). For 2D textures, s and t are used. Currently, the r coordinate is ignored (although it might have meaning in the future). The q co-ordinate, like w, is typically given the value 1 and can be used to create homogeneous co-ordinates. 纹理坐标命名strq与对象坐标进行区分 r 和 q 通常被设置成0和1
- We need to indicate how the texture should be aligned relative to the fragments to which it is to be applied. We can specify both texture co-ordinates and geometric co-ordinates as we specify the objects in the scene.
- The command to specify texture co-ordinates, `glTexCoord*()`, is similar to `glVertex*()`, `glColor*()` and `glNormal*()` - it comes in similar variations and is used in the same way between `glBegin()` and `glEnd()` pairs
- `glTexCoord{1234}{sifd}{v}()` 设置当前的纹理坐标

```
glBegin(GL_QUADS); // 开始画一个四边形

    // 1. 先告诉 OpenGL：我要用图片左下角（0, 0）
    glTexCoord2f(0.0f, 0.0f);

    // 2. 再告诉 OpenGL：这个坐标对应物体的左下角顶点
    glVertex3f(-1.0f, -1.0f, 0.0f);

    // 1. 先告诉 OpenGL：我要用图片右下角（1, 0）
    glTexCoord2f(1.0f, 0.0f);

    // 2. 再告诉 OpenGL：这个坐标对应物体的右下角顶点
    glVertex3f( 1.0f, -1.0f, 0.0f);<br/><br/>
    // ... 继续画上面两个点 ...

glEnd();
```

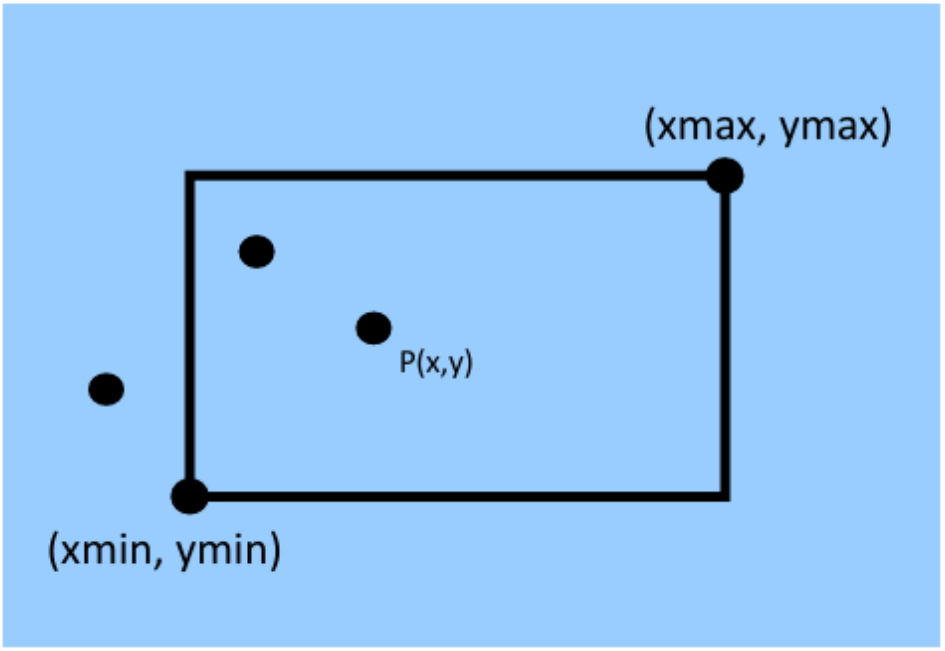
- OpenGL uses interpolation to find proper texels from specified texture coordinates
- OpenGL can generate texture co-ordinates automatically
  `glTexGen{ifd}{v}(GLenum coord, GLenum pname, GLint param)`
  coord: one of GL_S, GL_T, GL_R or GL_Q
  `pname` : generation modes(`GL_OBJECT_LINEAR` , `GL_EYE_LINEAR` , `GL_SPHERE_MAP`)

Perspective correction hint 透视矫正提示

纹理坐标和颜色插值要么在屏幕空间中线性插值 / 要么使用深度透视值（较慢）
`glHint(GL_PERSPECTIVE_CORRECTION_HINT,hint)`
which hint is one of `GL_DONT_CARE` , `GL_NICEST` , `GL_FASTEST`

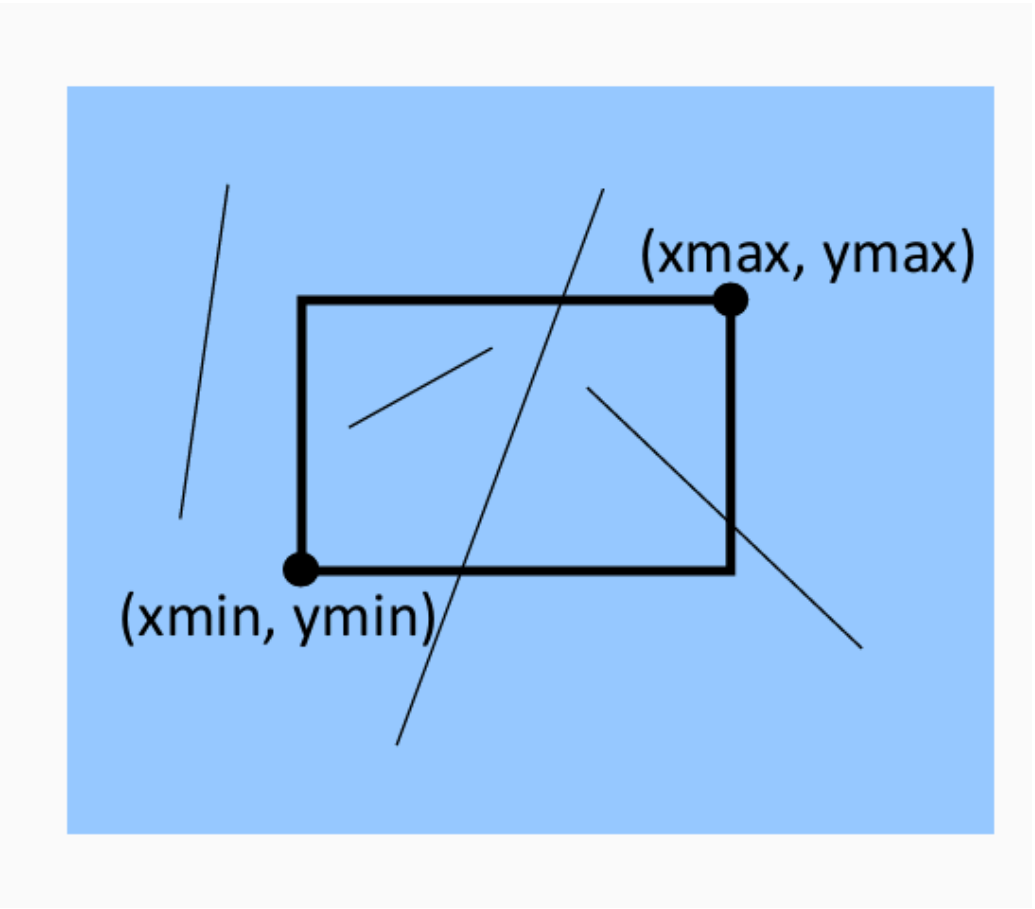
Lecture 11 Clipping 裁剪

- Clipping - remove objects or parts or parts of objects that are outside the clipping window 移除位于裁剪窗口之外的对象或对象的一部分
- Rasterization(scan conversion) -Convert high level object description to pixel color in the frame buffer 将高级对象描述转换为帧缓冲器中的像素颜色
- OpenGL do these for us -no explicit OpenGL functions needed for doing clipping and rasterization
- Rasterization meta algorithms
    - Consider 2 approaches to rendering a scene with opaque objects 渲染不透明物体场景的方法
    - For every pixel, determine which object that projects on the pixel is closest to the viewer and compute the shade of this pixel
        - Ray tracing paradigm 光线追踪范式
    - For every object, determine which pixels it covers and shade these pixels 对于每个物体，确定他覆盖了那些像素，并对这些像素着色
        - Pipeline approach
        - Must keep track of depths
- Clipping window vs viewport
  裁剪窗口时我们想要看到的内容，视口是它在输出位置上的查看位置
- Clipping primitives
    - Points
        - Determine whether a point (x, y) is inside or outside of the window



    - Lines
        - Determine whether a line is inside, outside or partially inside the window

- If a line is partially inside, we need to display the inside segment



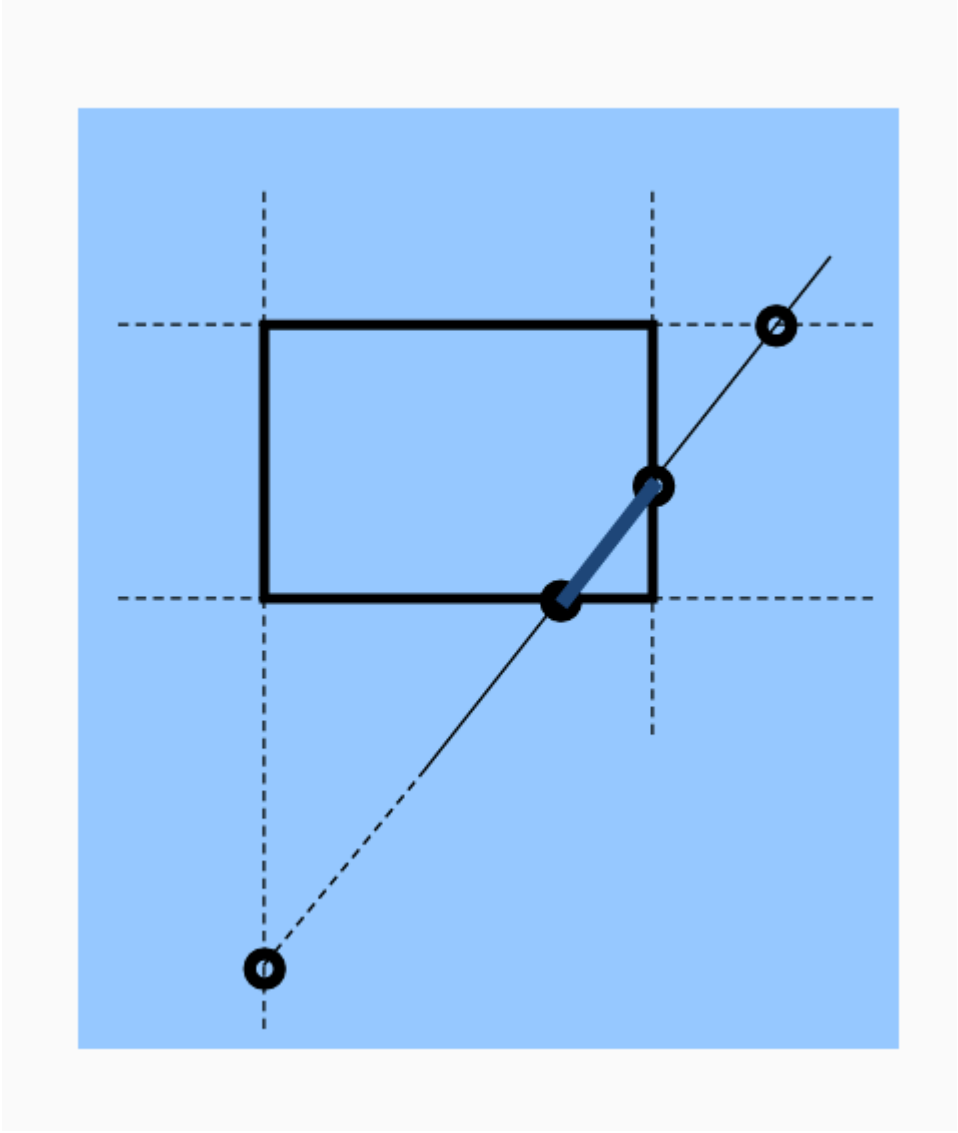
- Trivial cases:
  - Trivial accept(entirely within the clipping window) 完全在裁剪窗口内
  - Trivial reject(entirely outside one of the 4 clipping window edges) 完全在四个裁剪窗口之外
- Non-trivial cases
  - One point inside, and one point outside
  - Both points are outside, but not "trivially" outside 两个点都在外部但不是平凡的外部
  - Need to find the line segments that are inside 需要找到内部的线段
  - Compute the line-window boundary edge intersection 计算线与窗口交界处的焦点
  - There will be 4 intersections but only one or two are on the window edges
  - These 2 points are the end points of the desired line segment
- Polygons

Line clipping algorithms

Brute force clipping of lines(Simultaneous equation)

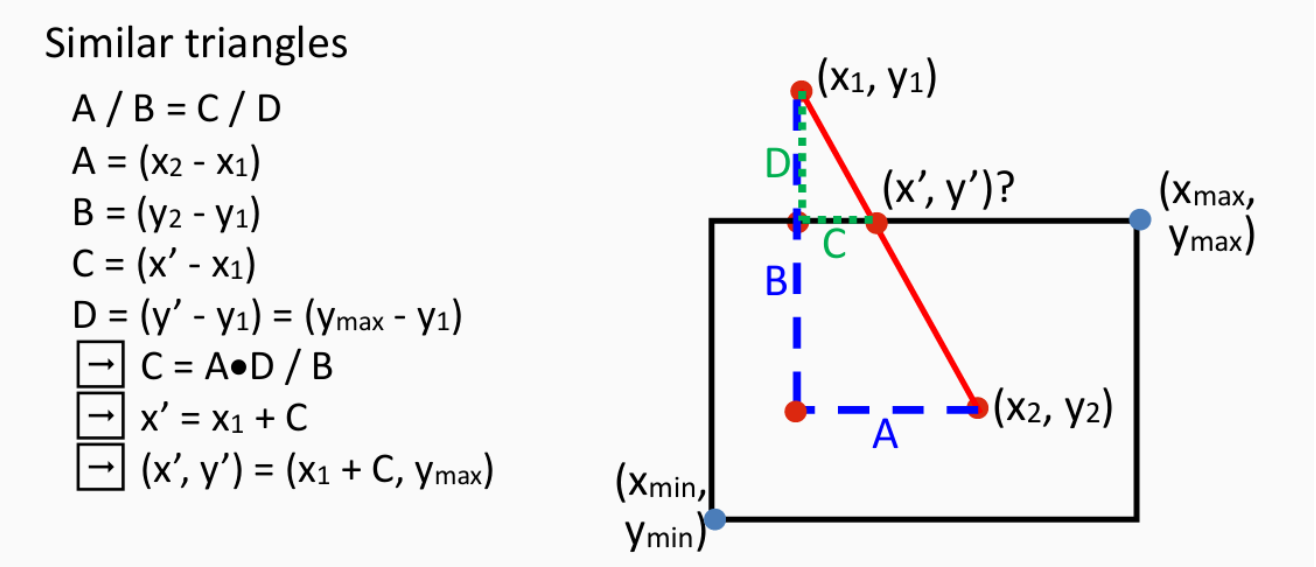
Brute force clipping: solve simultaneous equations using $y = mx + b$ for line and four clip edges

- Slope-intercept formula handles infinite lines only 斜截式只处理无限长的线
- Does not handle vertical lines



Brute force clipping of lines(Similar triangles)

Clip a line against an edge of the window, e.g. the top edge, so we need to find out x' and y' (which is y_{max})

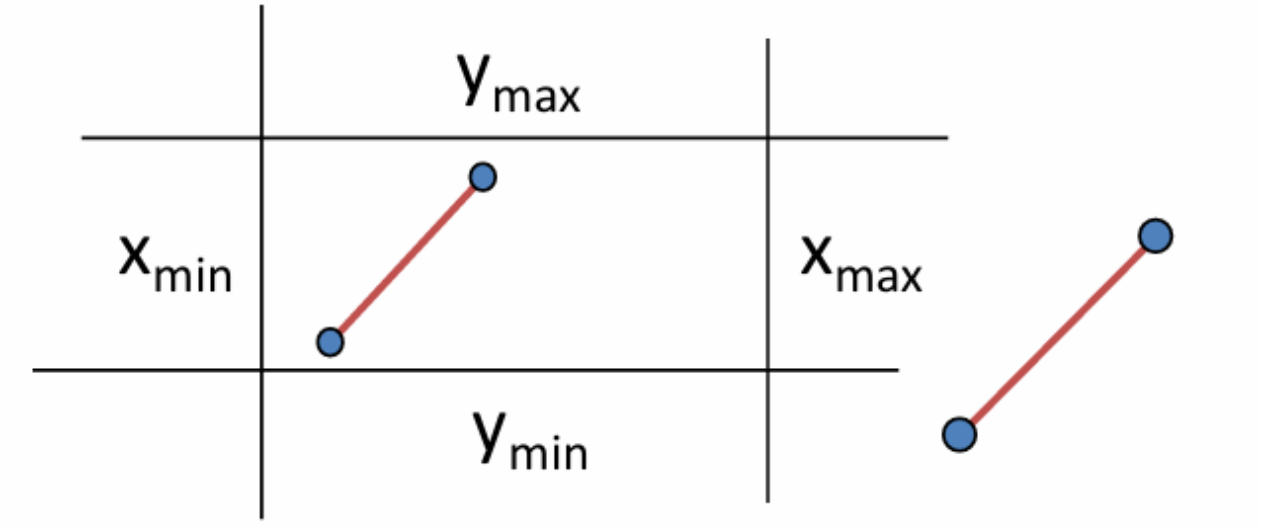


But the problem s too expensive(4 float point subtraction 1 floating point multiplication)

Cohen-Sutherland 2D line clipping

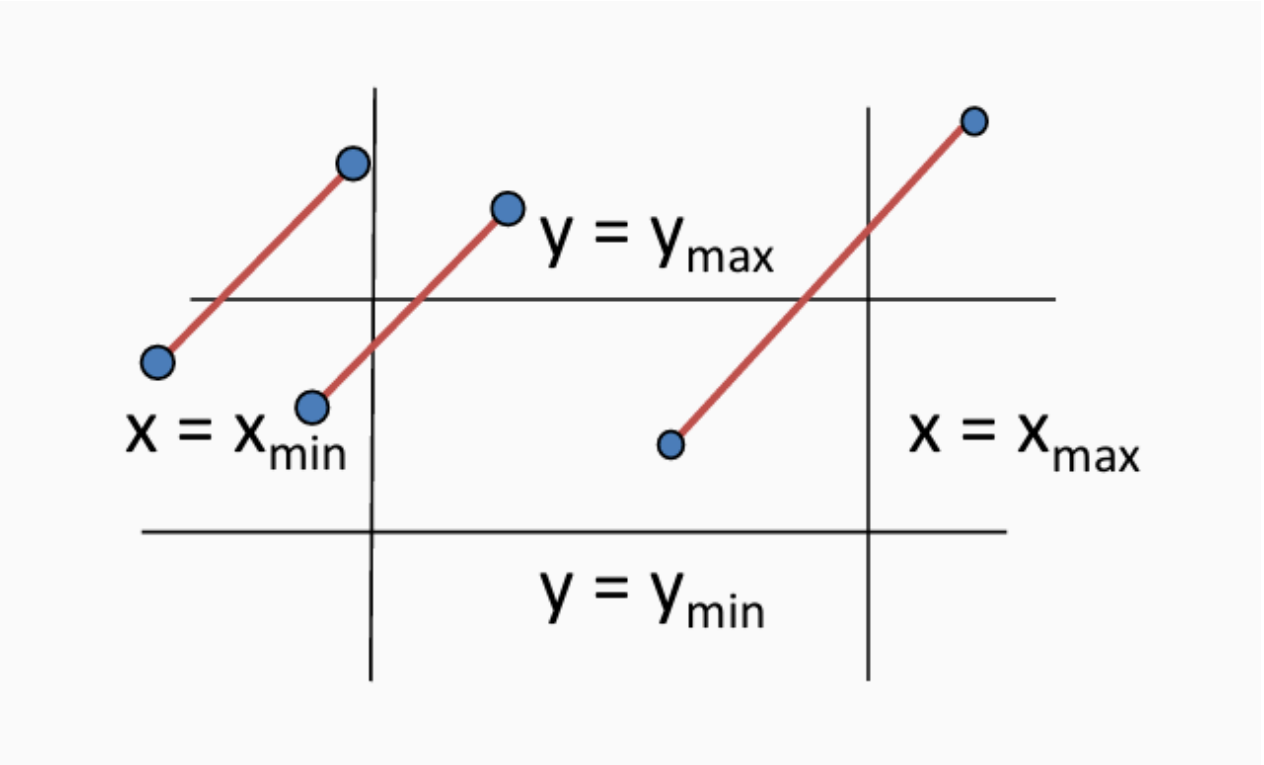
尽可能多的消除情况

- Creates an out code for each end point that contains location information of the point with respect to the clipping window 为每个端点创建一个编码 包含该点相对于裁剪窗口的位置信息
- Uses out codes for both end point to determine the configuration of the line with respect to the clipping window
- Computes new end points if necessary 必要时计算新端点
- Extends easily to 3D(using 6 faces instead of 4 edges)
- 线段两个端点都在四条线内部-按原样绘制线段
- 两个端点都在四条线外部 并且在一条线的同一侧 丢弃该线段



- 一个端点在内部 一个端点在外部 -必须进行一次求交（$x = x_{min}$）

- 两个端点都在外部 -可能有一部分在内部 必须进行至少一次求交



- Defining out codes
  - Split plane into 9 regions
  - Assign each a 4-bit out code(left right below above)
    $b_0 = 1$ if $x < x_{min}, 0$ otherwise
    $b_1 = 1$ if $x > x_{max}, 0$ otherwise
    $b_2 = 1$ if $y < y_{min}, 0$ otherwise
    $b_3 = 1$ if $y > y_{max}, 0$ otherwise
  - Computation of out code requires at most 4 subtractions
- When needed, clip line segment against planes: Use 6-bit out codes, 6 planes and 27 regions

Liang-Barsky (Parametric line formulation)

For the any point $P(x, y)$ in the line, its parametric formulation:

$$P(t) = P_0 + t(P_1 - P_0)$$

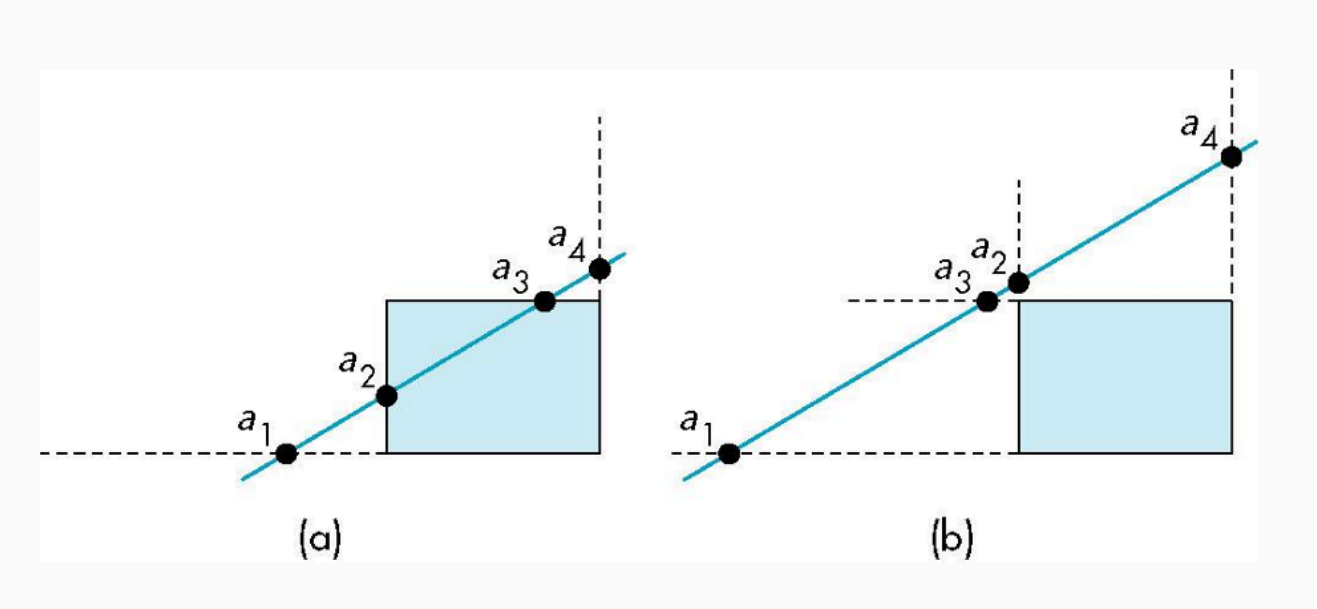
$$x = x_0 + t(x_1 - x_0)$$

$$y = y_0 + t(y_1 - y_0)$$

Consider the parametric form of a line segment

$$p(\alpha) = (1 - \alpha)p_1 + \alpha p_2, 0 \leq \alpha \leq 1$$

we can distinguish the cases by looking at the ordering of the α values where the line is determined by the line segment crossing the lines that determine the window



Can accept/reject as easily as with Cohen-Sutherland
Using values of α, we do not have to use the algorithm recursively as with the Cohen-Sutherland method

Polygon clipping

Plane-line intersections

If we write the line and plane equations in matrix form(where n is the normal to the plane and p_0 is a point on the plane), we must solve the equations

$$p(\alpha) = (1-\alpha)p_1 + \alpha p_2, 0 \leq \alpha \leq 1$$

$$n \cdot (p(\alpha) - p_0) = 0$$

For the α corresponding to the point of intersection, this value is

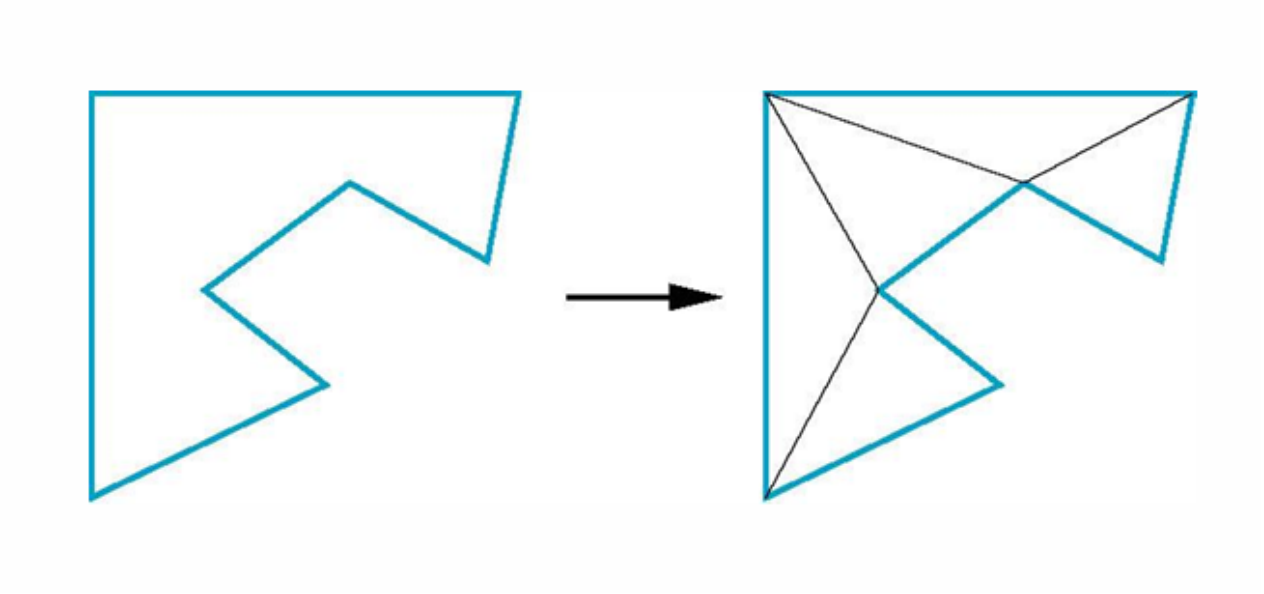
$$\alpha = \frac{n \cdot (p_0 - p_1)}{n \cdot (p_2 - p_1)}$$

The intersection requires 6 multiplications and 1 division

Convexity and tessellation 凸性与细分

- One strategy is to replace non-convex(concave) polygons with a set of triangular polygons 使用一组三角形多边形来替换非凸多边形

- Also makes fill easier



Clipping as a black box

we can consider line segment clipping as a process that makes in two vertices and produces either no vertices or the vertices of a clipped line segment 接收两个顶点 并输出要么没有顶点 要么是裁切后线段的顶点

Pipeline clipping of line segments

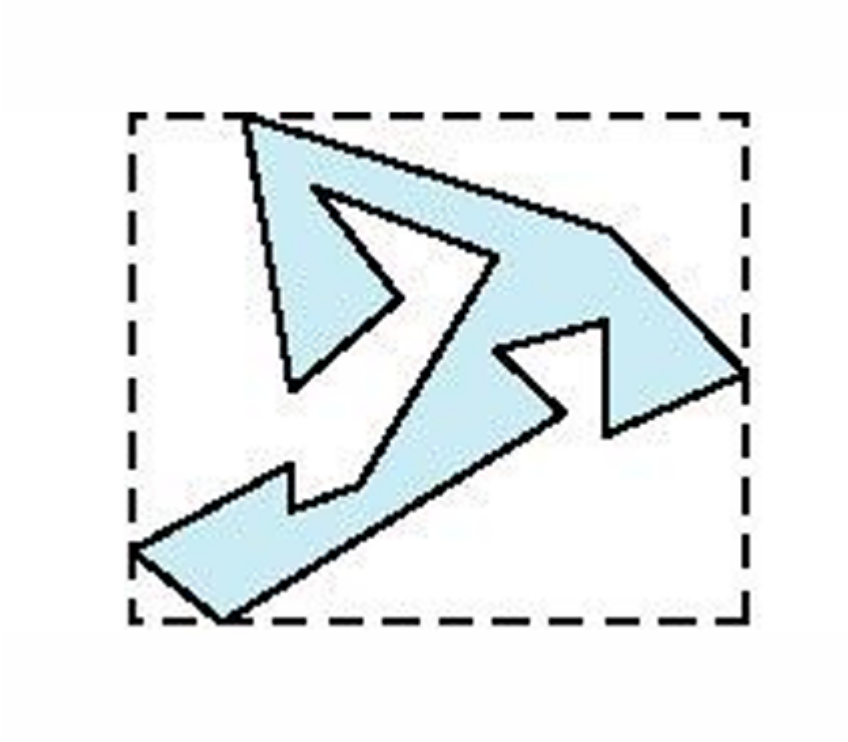
Clipping against each side of window is independent of other sides so we can use four independent clippers in a pipeline 针对窗口每条边的裁剪是独立于其他边的 因此在管线中使用四个独立的裁剪器
Three dimensions: add front and back clippers

Bounding boxes 包围盒

Rather than doing clipping on a complex polygon, we can use an axis-aligned bounding box or extent:

- smallest rectangle aligned with axes that encloses the polygon 包含多边形的最小与坐标轴对其的矩形

- Simple to compute: max and min of x and y



OpenGL functions

- Setting up a 2D viewport in OpenGL
  `glViewport(xvmin,yvmin,vpWidth, vpHeight);`
  如果不调用此函数 默认情况下会使用与现实窗口大象相同的视口
- Setting up a clipping-window in OpenGL

```
glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluOrtho2D(xwmin,xwmax,ywmin,ywmax);
glOrtho(xwmin,xwmax,ywmin,ywmax);

gluPerspective(vof,aspectratio,near,far);
glFrustum(xwmin,xwmax,ywin,ywmax,ear,far);
```

Lecture 12 Hidden-Surface Removal

- Clipping and hidden-surface removal
  - Clipping has such in common with hidden-surface removal
  - In both cases, we try to remove objects that are not seen to camera

    - Often we use visibility or occlusion testing early in the process to eliminate as many polygons as possible before going through the entire pipeline 可以在过程早期使用可见性和遮挡测试，在遍历整个管线之前消除尽可能多的多边形
- Display all visible surfaces, and do not display any occluded surfaces
    - Z-buffer is just one of hidden-surface removal algorithms

Hidden-surface removal algorithms

Object space algorithm 物体空间 vs Image space algorithms 图像空间

- Object space algorithm
    - It uses pairwise testing between polygons(objects)
    - Resize does not require recalculation 调整大小无需重新计算
    - Works for static scenes
    - May be different / impossible to determine
    - 从场景出发 向屏幕上看（关心物体本身的关系 所有物体的遮挡关系都在连续的数学空间里被计算）
- Image space algorithms
    - 确定每个像素处哪个物体可见
    - Resize requires recalculation
    - Works for dynamic scenes
    - 从屏幕出发 往场景里看（只关心离散的像素点 不管有多少个物体 只关心在屏幕上的一个像素上的前后关系）

Object-Space algorithm

Painter's algorithm

- Object-space algorithm
- Render polygons in the back to front order so that polygons behind others are simply painted over
- Sort surfaces/polygons by their depth(z value) 根据其深度对表面/多边形进行排序
    - Requires ordering of polygons first
    - $O(n \log n)$ calculations for ordering
    - Not evert polygon is either in front or behind all other polygons

Back-face culling 背面剔除

- Object-space algorithm

- Back-face culling is the process of comparing the position and orientation of polygons against the viewing direction v, with polygons facing away from the camera eliminated 将多边形的位置和方向与观察方向v 进行比较的过程 剔除了背向相机的多边形
- 这种消除最小化了隐藏面消除所涉及的计算开销
- Assume the object is a solid polyhedron 实行多面体
- Compute polygon normal n:
    - Assume a counter-clockwise vertex order
    - For a triangle (abc) : $n = (ab) \times (bc)$
    - If $90 \geq \theta \geq -90$ $v \cdot n \geq 0$, the polygon is a visbiable face
- For each polygon P_i
    - Find polygon normal n
    - Find viewer direction v
    - If $v \cdot n < 0$, then cull P_i
- 不适用于
    - Overlapping front faces due to multiple objects / concave objects
    - Non-polygonal models 非多边形模型
    - Non-closed objects 非封闭物体

Image Space algorithm

Z-buffer

- Image Space algorithm
- We now know which pixels contain which objects
- However since some pixels may contain 2 or more objects, we must calculate which of these objects are visible and which are hidden
- 图形硬件中通常使用这种方法
- In this algorithm, we set aside a two-dimensional array of memory(the Z-buffer) of the same size as the screen 分配一个与屏幕大小相同的二维内存数组 这是除了我们用于存储要现实的像素颜色值（帧缓冲器）之外的
- The Z-buffer will hold values which are depths 深度值
- Z缓冲被初始化 使每个元素具有远裁剪面的值
- Now for each polygon, we have a set of pixel values which the polygon covers 由一组该多边形覆盖的值
- For each of the pixels, we compare its depth (z-value) with the value of the corresponding element already stored in the Z-buffer: 对于每个像素 我们将其深度（z值）与Z缓冲中的存储的对应值进行比较
    - If this value is less than the previously stored value, the pixel is nearer the viewer than the previously encountered pixel; 如果该值小于当前存储的值 则该像素比之前遇到的像素更接近观察者

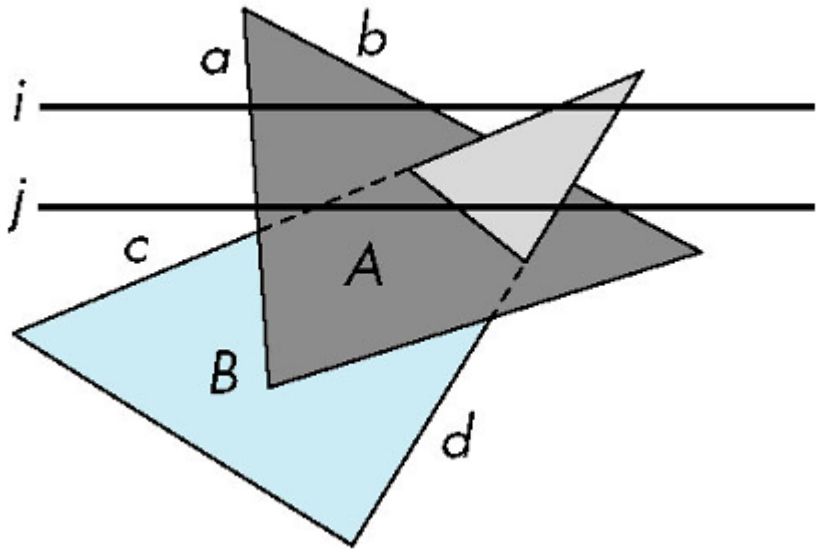
    - Replace the value of the Z- buffer with the z value of the current pixel, and replace the value of the color buffer with the value of the current pixel. 将Z缓冲的值替换为当前像素的Z值 并将颜色缓冲器的值替换为当前像素的值
- Repeat for all polygons in the image. 对图像中的所有多边形重复此过程
- The implementation is typically done in normalized co-ordinates so that depth values range from 0 at the near clipping plane to 1.0 at the far clipping plane 实现通常在规范化（归一化）坐标中进行 使深度从近裁剪面处的0到远裁剪面处的1.0
- 深度缓冲Z-buffer算法
    - 解决可见面判别问题 3D场景-2D屏幕图像时 决定哪些物体在前面（应该被显示）决定哪些物体在前面 哪些物体被挡在后面 计算机在渲染画面时 通常维护两张画布：颜色缓冲区Refresh Buffer/Frame Buffer 与 深度缓冲区 Z-buffer
    - Step 1 Initialize all depth(x, y) to 1.0 and $refresh(x, y)$ to all background color 在开始画图之前 先把深度缓冲区里的所有值都设为最大值（通常用1.0来代表最远处的极限）把颜色缓冲区涂成背景色（相当于清空屏幕）假设现在屏幕上所有点都距离我们要无限远
    - Step 2 For each pixel 算法核心循环 当计算机需要画一个物体（比如长方形或正方形）时，它会遍历该物体覆盖的每一个像素
        - 计算当前点的深度值（depth value z）算出这个物体在这个像素点上距离摄像机有多远
        - 比较深度（$z < depth(x, y)$）：
            - 它会拿当前计算出的新深度z 去和深度缓冲区里已经存在的就深度$depth(x, y)$做比较 数字越小 代表距离越近 新画的这个物体比原先在这里的物体更靠近物体
    - Step 3 Update z-buffer
        - $depth(x, y) = z$ 把深度缓冲区的值更新为这个更新 更近的距离
        - $refresh(x, y) = Is(x, y)$ 把屏幕上的颜色更新为这个新物体的颜色（I_s是经过光照 纹理计算后的颜色）
        - 房子 如果大于 说明新物体在就物体后面 被挡住了 所以什么都不做 直接丢弃这个像素
- Advantages
    - 最广泛使用的隐藏面消除算法
    - 一种图像空间算法 遍历场景并针对每个多边形而非像素进行操作
    - 在软件或硬件中更容易实现
- Memory requirements
    - 依赖于称为Z-buffer or depth buffer
    - Z缓冲与帧缓冲器frame buffer具有相同的宽度和高度
    - 每个单元格包含该像素位置的z值(距离观察者的距离)

Scan-line

If we work scan line by scan line, as we move across a scan line, the depth changes satisfy $a\Delta x + \Delta y + c\Delta z = 0$, noticing that the plane which contains the polygon is represented by $ax + by + cz + d = 0$

Along scan line
$\Delta y = 0$
$\Delta z = -(a/c) * \Delta x$
In screen space $\Delta x = 1$ Only computed once per polygon

- Worst case performance of an image-space algorithm is proportional to the number of primitives 图像空间算法的最坏情况性能与图元数量成正比
- Performance of z-buffer is proportional to the number of fragments generated by rasterization, which depends on the area of the rasterized polygons Z缓冲器的性能与光栅化生成的片段数量成正比 这取决于光栅化多边形的面积
  We can combine shading and hidden-surface removal through scan line algorithm

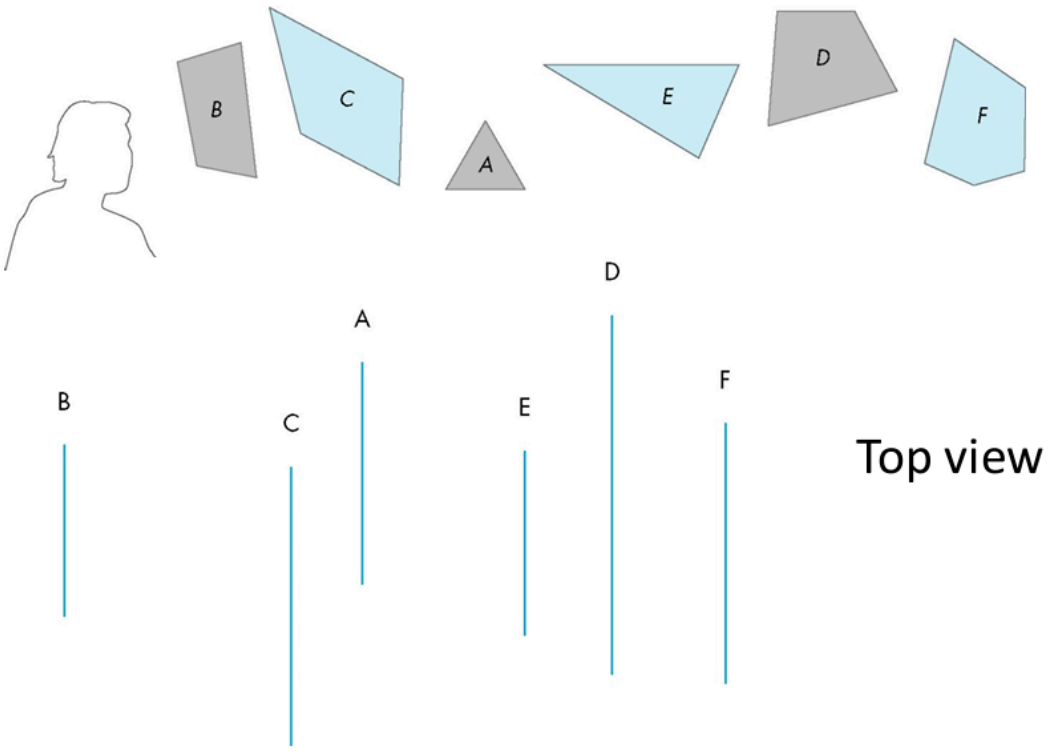


- Scan line i: no need for depth information, can only be in no or one polygon
- Scan line j: need depth information only when in more than one polygon 位于多个多边形内时
  Implementation: Need a data structure to store:
- Flag for each polygon(inside / outside) 每个多边形的标志
- Incremental structure for scan lines that stores which edges are encountered 用于存储遇到那些遍的扫描线的增量结构
- Parameters for plane 平面参数

BSP tree

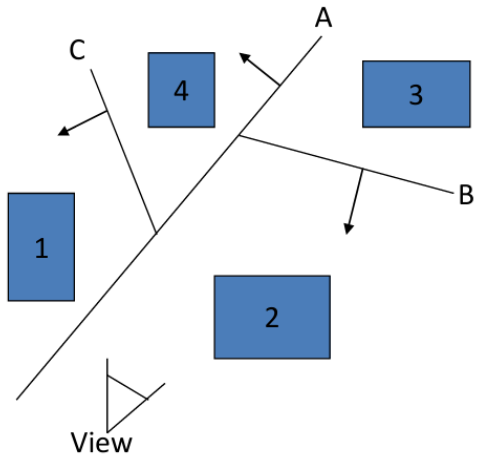
- In many real-time applications, such as games, we want to eliminate as many objects as possible within the application so that we can

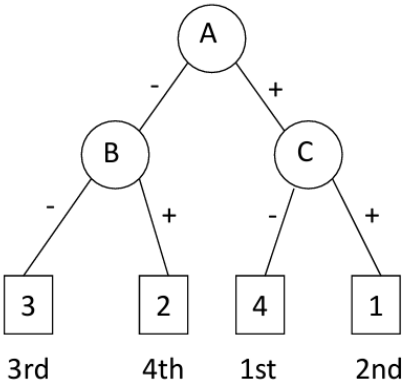
    - reduce burden on pipeline
    - reduce traffic on bus
- partition space with Binary Spatial Partition(BSP) Tree 二元空间分割树
- Consider 6 parallel planes. Plane A separates planes B and C from planes D,E and F.



- We can continue recursively
    - Plane C separates B from A
    - Plane D separates E and F
- We can put this information in a BSP tree
    - use the BSP tree for visibility(inside) and occlusion(outside) testing 使用BSP树进行可见性（内部）和遮挡（外部）测试
- The painter's algorithm for hidden-surface removal works by drawing all faces, from back to front 用于隐藏面消除的画家算法通过从后到前绘制所有面来工作
- Choose a polygon(arbitrary)
- Split the cell using the plane on which the polygon lies
    - May have to chop polygons into two if they intersect with the plane 如果多边形与平面相交 可能必须将多边形切成两部分（裁剪）
- Continue until each cell contains only one polygon fragment
- Splitting planes could be chosen in other ways, but there is no efficient optimal algorithm for building BSP trees
    - BSP trees are not unique - there can be a number if alternatives for the same set of polygons
    - Optimal means minimum number of polygon fragments in a balanced tree 最优意味着平衡树中多边形片段数量最少

- Observation: things on the opposite side of a splitting plane from the viewpoint cannot obscure things on the same side as the viewpoint 位于分割平面与视点相反一侧的物体不能遮挡与视点同侧的物体
    - 你（viewpoint）在墙（splitting plane）的这边 树在墙的那边 树绝对不可能挡住墙 只有可能是墙挡住树
- The rendering is recursive of the BSP tree
- At each node(for back to front rendering):
    - Recurse down the side of the sub-tree that does not contain the viewpoint -Test viewpoint against the spilt plane to decide the polygon 递归遍历不包含视点的子树一侧 测试视点与分割平面决定多边形
    - Draw the polygon in the splitting plane - Paint over whatever has already been drawn
    - Recurse down the side of the tree containing the viewpoint 递归遍历包含视点的树的一侧





- explanation：
    - 现在我在路口A 眼睛在A的负侧（-) 决定既然眼睛在负侧 就先画完正侧的东西 动作 展示不管B那边 先处理C
    - 到达C这条线 C的箭头指向区域1 眼睛对于C来说是在正侧的 C想象成无限延伸长的 动作是把区域4画完 再画区域1
    - 然后画A的正侧的 来到B这条线 既然眼睛在正侧 区域2 先把负侧的区域3画完 再来画区域2

OpenGL functions for hidden-surface removal

OpenGL uses the z-buffer algorithm that saves depth information as objects / triangles are rendered so that only the from objects appear in the image 该算法在渲染物体/三角形时保存深度信息 以便只有前面的物体出现在图像中

OpenGL Z-buffer functions

- The algorithm uses an extra buffer, the z-buffer, to store depth information as geometry travels down the pipeline
- It must be
  - requestes in `main()` `glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH)`
  - enabled in `init()` `glEnable(GL_DEPTH_TEST)`
  - cleared in the display callback 在显示回掉中清除 `glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)`
- The depth values are normalized in the range $[0.0, 1.0]$ But to speed up the surface processing, a maximum depth value, maxDepth, can be specified between 0.0 and 1.0 `glClearDepth(maxDepth)`
- Projection co-ordinates are normalized in the range $[0.0, 1.0]$, and the depth values between the near and far clipping planes respectively 近裁剪平面和远裁剪平面之间的深度值进一步在范围内归一化 其中0.0和1.0 分别对应于近裁剪平面和远裁剪平面
- The function below allows adjustment of the clipping planes `glDepthRange(nearNormDepth,farNormDepth` 其中两个参数默认值分别是0.0和1.0
- Another function for extra options is `glDepthFunc(testCondition)` `testCondition` can have the following values: `GL_LESS,GL_GREATER,GL_EQUAL,GL_NOTEQUAL,GL_LEQUAL,GL_GEQUAL,GL_NEVER` (no points are processed) and `GL_ALWAYS` (all points are processed) for the application
- The status of the depth buffer can be set to read-only or read-write `glDepthMask(writeStatus)`
  - When `writeStatus=GL_FALSE`, the write mode in the depth buffer is disabled and only retrieval of values for comparison is possible 深度缓冲器中的写入模式被禁用 智能检索值进行比较
  - It is useful when a complex background is used with display of different foreground objects
    - 将背景存储在深度缓冲器中 禁用写入模式以处理前景 允许生成一系列具有不同情境物体或具有一个物体在不同位置的帧用于动画序列 在显示透明物体时很有用

OpenGL functions

- OpenGL basic library provides functions for back-free removal and depth-buffer visibility testing
- In addition, there are hidden-line removal functions for wireframe display, and scenes can be displayed with depth cueing

Face-culling functions

- Back-face removal(and back-face culling) is accomplished with the functions
  `glEnable(GL_CULL_FACE)`
  `glCullFace(GLenum)`
  The parameter mode includes `GL_BACK,GL_FRONT and GL_FRONT_AND_BACK` So front faces can be removed instead of back faces or both front and back faces can be removed
- The default value for `glCUllFace()` is `GL_BACK`. Therefore, if `glEnable(GL_CULL_FACE)` is called to enable face culling without explicitly calling `glCullFace()`, back faces in the scene will be removed

wireframe surface-visibility functions

A wireframe display of an object can be generated with
`glPolygonMode(GL_FRONT_AND_BACK,GL_LINE)`
both visible and hidden edges are display

Depth-cueing functions

The brightness of an object is varied as a function of its distance to the viewing position 对象的亮度随其观察位置的距离而变化

```
glEnable(GL_FOG)
glFOG{if}[v](GL_FOG_MODE,GL_LINEAR) //linear depth function in [0.0,1.0]
glFog{if}[v](GL_FOG_START,minDepth) //Specifies a different value for dmin
glFog{if}[v](GL_FOG_END,maxDepth) // Specifies a different value for dmax
```