**COMP 390 Notes Jingze Dai**

Student Number: 3491646

**Section 7.1**

**Translation**: Moving an object

1. Moving method: current coordinate + update coordinate amount = new coordinate.

**Rotation**: Rotate an object

1. Rotating method: With a rotation center, move objects with given angles.

**Scaling**: Making an object larger or smaller

1. Scaling method: coordinates are multiplied by given positive numbers, and

**Section 7.2**

**All of those transformations are made by vector updating and changing.**

**Homogeneous Coordinates:** coordinate system created from different vectors, those vectors are not zero and they are not parallel. (i.e. (X, Y, Z) 3d coordinate system)

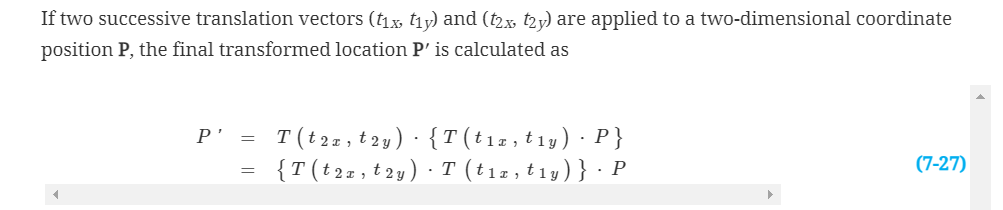
**Question**: Let T be a 2 x 2 transformation matrix, and R be a 2 x 2 rotation matrix. Does T \* R always equal to R \* T? In other words, will a different sequence of transformations yield the same result?

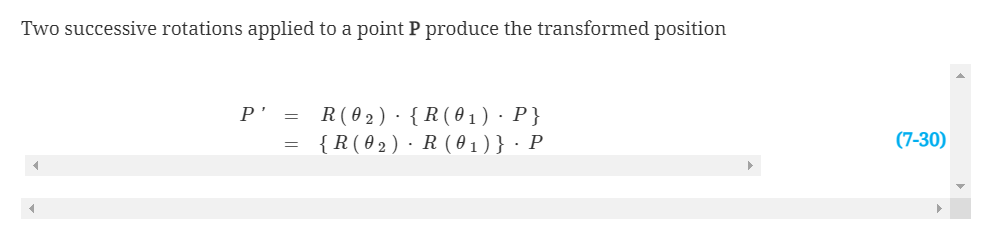
**Answer**: No

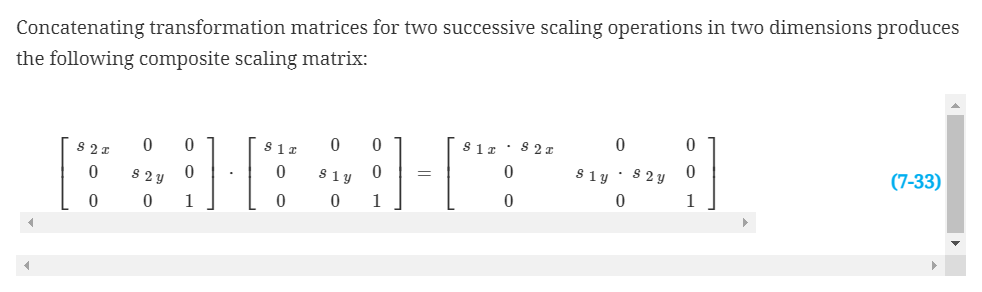
**Section 7.3**

**Inverse Transformation**: giving negative sign on coordinates.

**Section 7.4**







**Section 7.8**

Transformation between 2 different 2D coordinate system

1. Translate so that the origin (x0, y0) of the x′y′ system is moved to the origin (0, 0) of the xy system.
2. Rotate the x′ axis onto the x axis.

**Section 7.9**

**Translation, rotation and scaling**

glTranslate\* (tx, ty, tz);

glRotate\* (theta, vx, vy, vz);

glScale\* (sx, sy, sz);

**Set matrix system (coordinate system) and model view mode**

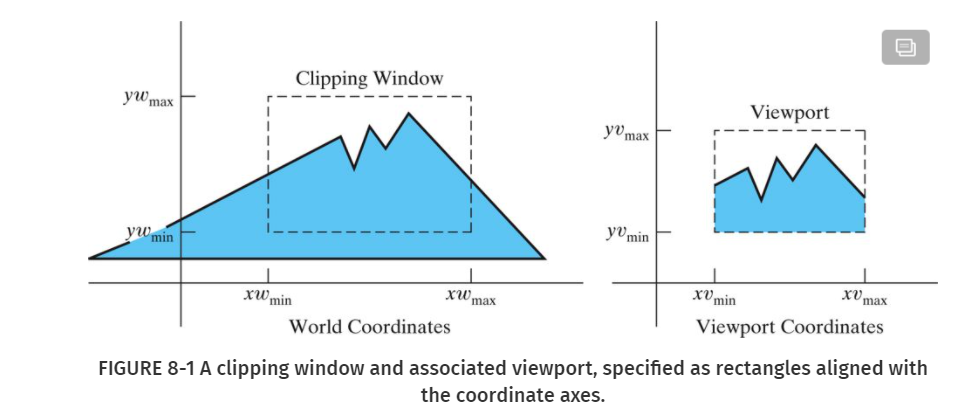
glMatrixMode (GL\_MODELVIEW);

**Assign the identity matrix to current matrix**

glLoadIdentity ();

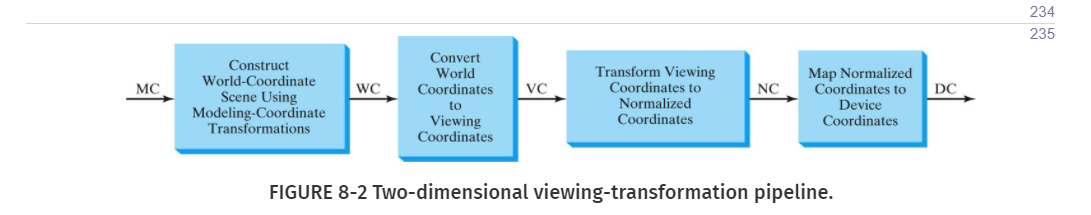
**Section 8.1**

**Clipping window**: the window with viewing contents in the scene. (To achieve a particular viewing effect in an application program)



**Section 8.2**

To achieve a particular viewing effect in an application program, we could design our own clipping window with any shape, size, and orientation we choose.



**Section 8.3**

**Question**: What are the steps in a 2D viewing transformation pipeline?

1. Scale the clipping window to the size of the viewport using a fixed-point position of (XWmin, YWmin).
2. Translate (XWmin, YWmin) to (XVmin, YVmin).

**Question**: What is a clipping window?

**Clipping window**: the window with viewing contents in the scene. (To achieve a particular viewing effect in an application program)

**Question**: What is a normalized viewport?

**Normalized viewport**: a polygon viewing (visible) region in computer graphics with normalized coordinates (between 0 and 1) with purpose: To illustrate the general procedures for the normalization and viewport transformations.

**Section 8.4**

Define a two-dimensional clipping window

gluOrtho2D (xwmin, xwmax, ywmin, ywmax);

Specify the viewport parameters

glViewport (xvmin, yvmin, vpWidth, vpHeight);

Resize the window to 400 x 400 pixels.

glutInitWindowSize (400, 400);

Enlarge the window using the mouse. Comment out the glViewport comment in the routine reshape(). Try enlarging the window again. What will happen?

After commenting out the glViewport command, the size of the square does not enlarge in proportion to the size change. (Viewport has the same size as before and we have the same part of visible area as before, only the initial viewing content is enlarged)

**Section 19.1**

**Electromagnetic Spectrum**: a special physics experiment that decomposes natural light, and its output is a line with different colors. Lights have different frequencies and power so they have different colors.

**Brightness**: total energy of lights

**Luminance**: quantified brightness (compared with brightness)

**Chromaticity**: purity of colors, color characteristics: purity and dominant frequency

**Section 19.2**

**Primary Colors (RGB)**: Colors use to form other colors.

**Section 19.4**

Why is the RGB model referred to as an additive model?

**C(λ) = XX + YX + ZX**. It means that different colors are produced by 3 basic primary colors: Red, Green and Blue.

**Section 19.5**

**YIQ color model**: NTSC color encoding for forming the composite video signal

1. Luminance (brightness) information is conveyed by the Y parameter
2. Chromaticity information (hue and purity) is incorporated into the I and Q parameters
3. Parameter I contain orange-cyan color information that provides the flesh-tone shading
4. Parameter Q carries green-magenta color information
5. Black-and-white television monitors use only the Y signal

**Section 19.6**

**CMY color model**: A subtractive color model can be formed with the three primary colors cyan, magenta, and yellow. Each of them is the combination of 2 RGB primary colors. Subtract their partial effects will from different colors.

**Section 5.3**

**Set current color components**

glColor\* (colorComponents);

glColor3fv (colorArray);

glColor3ub (0, 255, 255);

**Apply and Turn off color blending**

glEnable (GL\_BLEND);

glDisable (GL\_BLEND);

**Apply the clear color to background**

glClear (GL\_COLOR\_BUFFER\_BIT);

**Section 9.1 && 9.2 && 9.3**

What are the differences between a 2D transformation matrix and a 3D transformation matrix?

1. Positions use 3 coordinates to represent (instead of 2)
2. Calculation array size becomes 4 \* 4 instead of 3 \* 3.
3. Height effects and other 3d effects need to be considered.

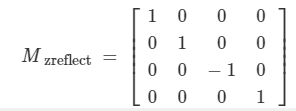
**Section 9.4**

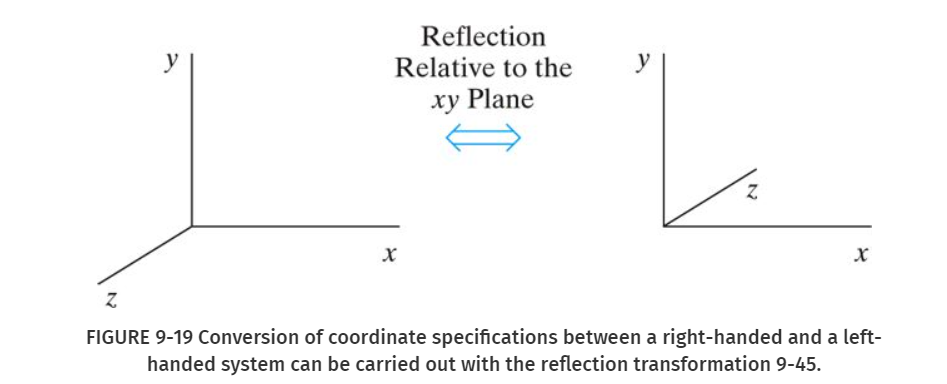
**Question**: Let T be a transformation matrix, and R be a rotation matrix. Is T \* R always interchangeable with R \* T?

**No (The same as 2D transformation)**

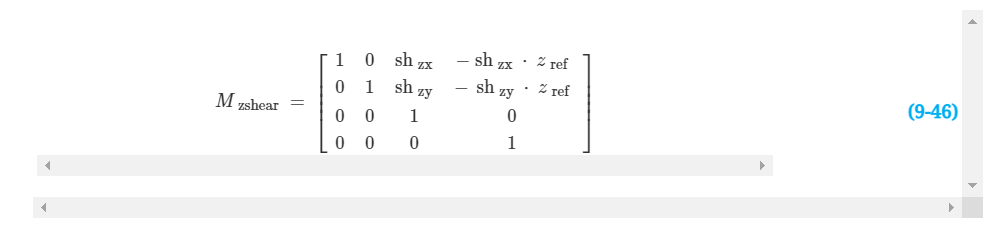
**Section 9.5**

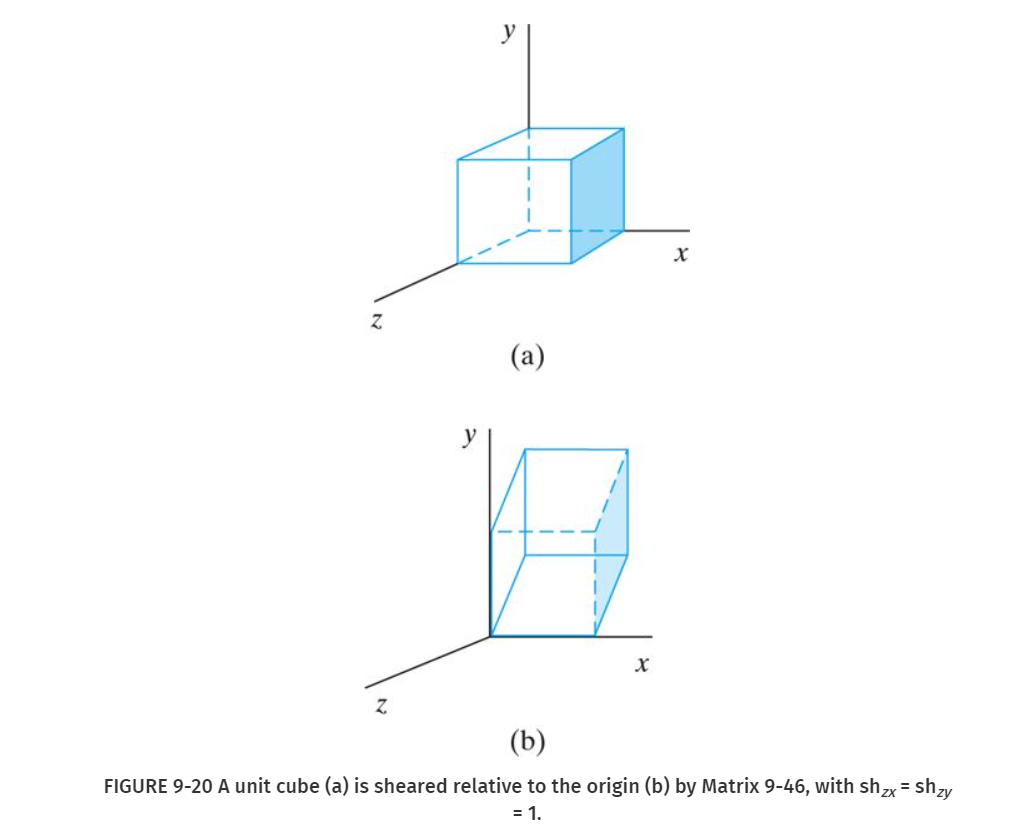
**Reflection transformation matrix**: Compared with initial coordinate systems, reflection transformation matrix has 2 dimension the same but 1 dimension reverse.





**Shear matrix**: A general z-axis shearing transformation relative to a selected reference position is produced with the following matrix:





**Section 9.8 and Section 9.9**

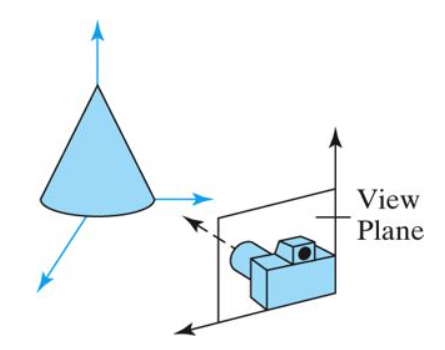
Copy the current matrix at the top of the active stack and store that copy in the second stack position (duplicate matrices at the top two positions of the stack).

**glPushMatrix();**

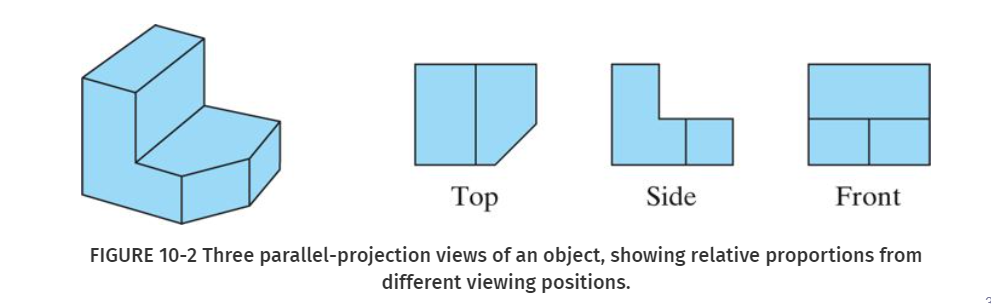
Destroys the matrix at the top of the stack, and the second matrix in the stack becomes the current matrix.

**glPopMatrix();**

**Section 10.1 && 10.2 && 10.3**



**Projections**: generating a view of a three-dimensional scene is to project points to the view plane along converging paths



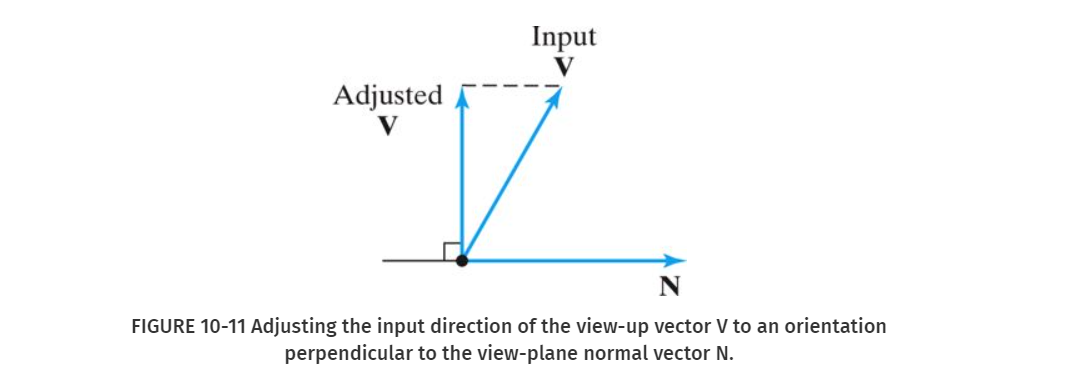
**Clipping planes**: imaginary planes located at two particular distances from the camera along the camera's sight line.

**View Volume**: a three-dimensional clipping region is established.

**View-Plane Normal Vector**: Vector perpendicular and meets the center of viewing plane (It shows the direction of observation)

**Section 10.4**

Usually, V is defined by selecting a position relative to the world-coordinate origin, so that the direction for the view-up vector is from the world origin to this selected position. Because the view-plane normal vector N defines the direction for the zview axis, vector V should be perpendicular to N. But, in general, it can be difficult to determine a direction for V that is precisely perpendicular to N. Therefore, viewing routines typically adjust the user-defined orientation of vector V, as shown in Figure 10-11, so that V is projected onto a plane that is perpendicular to the view-plane normal vector.



**Question**: Are world coordinates and viewing coordinates always the same?

World coordinates and viewing coordinates are not always the same.

**Section 10.5 && 10.6**

**Section 10.7**

**Section 10.8**

**Section 10.11**

**Section 11.1 && 11.2**

**Symbols**: Component parts of design system are displayed as geometric structures

**Instance**: examples of symbols.

**Modules**: consist of many basic objects, have more complex structure.

**Section 11.3 && 11.4 && 11.5**

To construct a graphical model, we apply transformations to the local-coordinate definitions of symbols to produce instances of the symbols within the overall structure of the model. Transformations applied to the modeling-coordinate definitions of symbols to give them a particular position and orientation within a model are referred to as modeling transformations. The typical transformations available in a modeling package are translation, rotation, and scaling, but other transformations might also be used in some applications.

**Section 13.2**

**Draw wired and solid tetrahedron.**

glutWireTetrahedron();

glutSolidTetrahedron();

**Draw wired and solid cube.**

glutWireCube (edgeLength);

glutSolidCube (edgeLength);

**Draw wired and solid 12-sided polygon.**

glutWireDodecahedron();

glutSolidDodecahedron();

**Draw wired and solid 20-sided polygon.**

glutWireIcosahedron();

glutSolidIcosahedron();

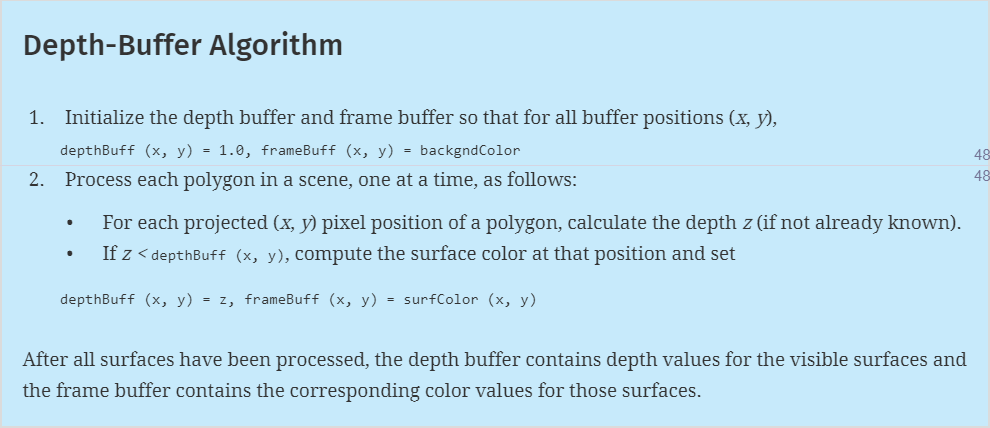
**Section 16.1 && 16.2 && 16.3 && 16.4**

**Back Face (Object-space method)**: locate back-faces (invisible parts) of polygons.

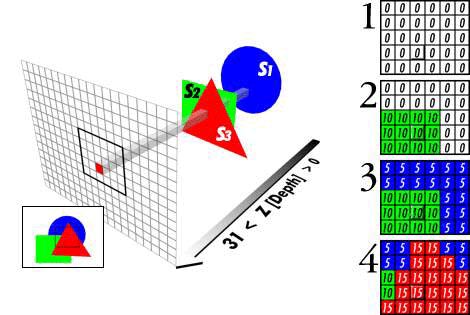
If point (x, y, z) is behind a polygon surface, Ax + By + Cz + D < 0

If Vview is a vector in the viewing direction from our camera position, then a polygon is a back face if Vview \* N > 0

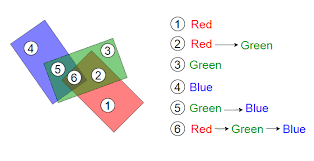
**Z-Buffer (Depth Buffer)**: image-space approach which compares surface depth values throughout a scene for each pixel position on the projection plane



A depth buffer, also known as a Z-Buffer, is a type of data buffer used in computer graphics used to represent depth information of objects in 3D space from a particular perspective. Depth buffers are an aid to rendering a scene to ensure that the correct polygons properly occlude other polygons.



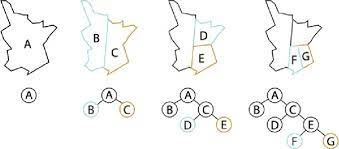
**A-Buffer (extension of Depth-Buffer)**



**Section 16.5 – Section 16.10**

Scan-Line Method: This image-space method for identifying visible surfaces computes and compares depth values along the various scan lines for a scene.

BSP-Tree: A binary space-partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces into the frame buffer from back to front, as in the painter’s algorithm. The BSP tree is particularly useful when the view reference point changes, but the objects in a scene are at fixed positions. (Using BSP Trees to divide areas into smaller and smaller parts, and put all of those parts in painting buffer)



**Section 16.14**

**OpenGL depth-buffer visibility-detection routines are activated**

glEnable(GL\_DEPTH\_TEST);

**Deactivate the depth-buffer routines**

glDisable(GL\_DEPTH\_TEST);

**Section 17.1 && 17.2**

**Light Sources**

1. Point Light Sources
2. Infinitely Distant Light Sources
3. Directional Light Sources and Spotlight Effects
4. Extended Light Sources and the Warn Model

**Light Intensity**: quantity of visible lights in per unit of area.

**Surface Rendering**: a procedure for applying a lighting model to obtain pixel colors for all projected surface positions.

**Diffuse Reflection**: scattered lights on surfaces that are rough or grainy tend to scatter the reflected light in all directions (general reflection on all areas of intersections)

**Specular reflection**: some of the reflected light is concentrated into a highlight, or bright spot (small area or point brighter lighting parts)

**Section 17.3**

In computer graphics, light usually consists of multiple components. The overall effect of a light source on an object is determined by the combination of the object's interactions with these components. The three primary lighting components (and subsequent interaction types) are diffuse, ambient, and specular.

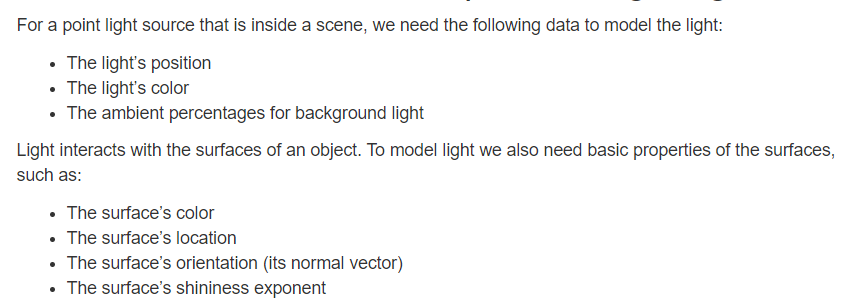
**Ambient lighting**: a general level of illumination that does not come directly from a light source

**Diffuse lighting**: direct illumination of an object by an even amount of light interacting with a light-scattering surface

**Specular lighting**: gives objects shine and highlights

What are the parameters that define ambient, diffuse, and specular lighting, respectively?

1. Light source direction vector: vector that shows lighting direction.



**Section 17.11**

**Enable and disable lighting**

glEnable(GL\_LIGHTING);

glDisable(GL\_LIGHTING);

**Set material effects**

surfEmissionColor [ ] = {0.8, 0.8, 0.8, 1.0};

glMaterialfv (GL\_FRONT, GL\_EMISSION, surfEmissionColor);

**Total ambient illumination** = intensity of the ambient light \* ambient light reflection coefficient.

**Total diffuse illumination** = intensity of the diffuse light source \* proportion of diffuse light reflected by a material \* angle between the beam of light and the surface normal

**Total specular illumination** = attenuation factor \* intensity of the specular light source \* specular reflection coefficient of the material ( cos^n a )

(Simulates a gentle fall-off of specular reflection when n is small, and simulates a sharp fall-off when n is large)

**Section 17.4**

How does the depth sorting visibility algorithm handle a group of transparent objects?

A depth-sorting visibility algorithm can be modified to handle transparency by first sorting surfaces in depth order, then determining whether any visible surface is transparent. If it is, its reflected surface intensity is combined with the surface intensity of objects behind it to obtain the pixel intensity at each projected surface point.

Transparent Objects: objects that can be seen behind objects.

**Section 17.5**

What general strategies are used to simulate atmospheric effects?

The hazy-atmosphere effect is often simulated with an exponential attenuation function such as fatmo(d) = e−ρd

**Section 18.1 && 18.2**

**Texture Mapping**: map patterns onto the geometric description of the object

**Volume texture patterns**: designate a set of colors for positions throughout a three-dimensional region of space

**MIP maps**: reduction patterns that have half size of previous patterns. (mipmaps (also MIP maps) or pyramids are pre-calculated, optimized sequences of images, each of which is a progressively lower resolution representation of the previous.)

**Procedural Texturing Methods**: use a procedural definition for the color variations that are to be applied.

**Section 5.2**

**Greyscale**: Because color capabilities are now common in computer-graphics systems, we use RGB color functions to set shades of gray, or grayscale, in an application program. When an RGB color setting specifies an equal amount of red, green, and blue, the result is some shade of gray. Values close to 0 for the color components produce dark gray, and higher values near 1.0 produce light gray. Applications for grayscale display methods include enhancing black-and-white photographs and generating visualization effects.

**Section 17.9**

**Halftones**: Continuous-tone photographs are reproduced for publication in newspapers, magazines, and books with a printing process called halftoning, and the reproduced pictures are called halftones.

**Halftone approximation patterns**: rectangular pixel regions used to simulate halftone reproductions.

**Dither noise**: Random values added to pixel intensities to break up shapes of textures.

**Ordered dither**: generates intensity variations with a one-to-one mapping of points in a scene to pixel positions using a dither matrix Dn to select an intensity level.

What are the general steps of the dithering technique?

1. Create different sizes’ dots.
2. map grey values onto bi-level values
3. fill in the area with ‘clumps’ of intensity

**Section 17.10**

What are the general assumptions of flat surface rendering?

1. The polygon is one face of a polyhedron and not a section of a curved-surface approximation mesh.
2. All light sources illuminating the polygon are sufficiently far from the surface that N · L and the attenuation function are constant over the area of the polygon.
3. The viewing position is sufficiently far from the polygon that V · R is constant over the area of the polygon.

What are the general steps of Gouraud surface rendering?

1. Determine the average unit normal vector at each vertex of the polygon.
2. Apply an illumination model at each polygon vertex to obtain the light intensity at that position.
3. Interpolate the vertex intensities linearly over the projected area of the polygon.

What are the general steps of Phong shading?

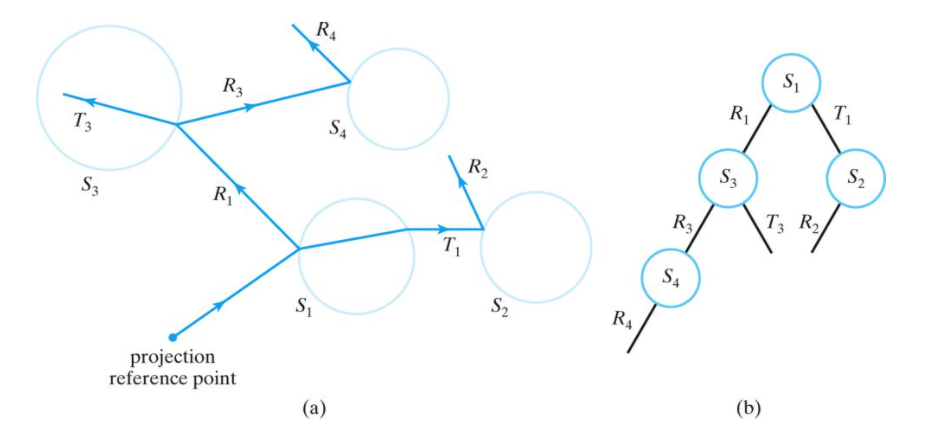
1. Determine the average unit normal vector at each vertex of the polygon.
2. Interpolate the vertex normal linearly over the projected area of the polygon.
3. Apply an illumination model at positions along scan lines to calculate pixel intensities using the interpolated normal vectors.

**Section 21.1**

**Ray Tracing**: the generalization of the basic ray-casting procedure.

**Secondary ray**: For a transparent surface, we also send a ray through the surface in the refraction direction. The reflection and refraction rays are referred to as secondary rays.

**Ray-tracing tree**



The parameters required to implement the basic recursive algorithm are: **depth, reflection, and refraction coefficients, calculation of intersections, light sources, and properties of materials.**

What is the general strategy for calculating ray – object intersections?

1. Use a beginning point and a vector to represent light path. (a linear equation)
2. With different passed objects, calculate new secondary rays on them.
3. Record intersection points.

**Space-subdivision**: a kind of reduced intersection calculation: enclose an entire scene within a cube, then we successively sub-divide the cube until each subregion (cell) contains no more than a preset maximum number of surfaces. Use those unit cubes to represent possible surfaces and use them in calculations.

**Light buffer technique**: An object is centered on each point light source, and each side of the cube is partitioned using a grid of squares. A sorted list of objects that are visible to the light through each square is then maintained by the ray tracer to speed up processing of shadow rays. As a means for determining surface illumination effects, a square for each shadow ray is computed and the shadow ray is then processed against the list of objects for that square.

**Super sampling**: light antialiasing: uses multiple, evenly spaced rays (samples) over each pixel area. (pixel is treated as a finite square area instead of a single point)

**Adaptive sampling**: light antialiasing: uses multiple, unevenly spaced rays (samples) over each pixel area. (pixel is treated as a finite square area instead of a single point)

**Distribution ray tracing**: a stochastic sampling method that randomly distributes rays according to the various parameters in an illumination model.

What do you need to calculate intersection of ray cast from the viewer with the plane?

1. The equation of the plane
2. The surface normal of the plane
3. The position and visual characteristics of the light sources
4. An algorithm to calculate whether the intersection point lies inside or outside of the plane.

**Pixel Method**: The basic idea is to treat the entire viewing window as a 2D map of pixels. A ray will be cast from the viewer to the each of the pixels, and then toward the scene to the first intersection. After the visual properties of all pixels are calculated, the pixel map will be rendered using the glDrawPixels command.

Routine: Recursive Ray Tracer

Return value: Colour Contribution

Parameters: Incident ray, cumulative recursive ray tracer calls

If termination condition is met // termination conditions:

Return “no contribution” // maximum recursive calls, or

// too far away

Else

Find the nearest intersection

Cast a ray from the intersection to the light sources

For each light source

Find if there is any intersection point in between

If intersection exists

Calculate lighting intensity with shadow

Else

Calculate lighting intensity w/o shadow

Calculate local color

Adjust for aliasing effect

If the object at the intersection point is reflective

Calculate reflective ray

Call Recursive Ray Tracer to get reflective color

If the object at the intersection point is refractive

Calculate refractive ray

Call Recursive Ray Tracer to get refractive color

Color at the intersection point =

(Local color \* local color contribution percentage) +

(Reflective color \* reflective color contribution percentage) +

(Refractive color \* refractive color contribution percentage)

Return Color at the intersection point

**Chapter 14.1**

**Spline curve**: curves drawn by spline (curve’s position and shapes are controlled by a group of points)

**Spline surface**: curves drawn by spline (2 sets of spline curves)

**Chapter 12.10**

**Real-time animation**: each stage of the sequence is viewed as it is created. Thus, the animation must be generated at a rate that is compatible with the constraints of the refresh rate.

**Frame-by-frame animation**: each frame of the motion is separately generated and stored. Later, the frames can be recorded on film, or they can be displayed consecutively on a video monitor in “real-time playback” mode

**Chapter 17.11**

What are the parameters used to set a spotlight?

1. Spotlight direction vector
2. Cone angle
3. angular-attenuation exponent (How does light increase and decrease)

**Chapter 22.3**

**Shaders**: Shaders are small program fragments that are loaded into OpenGL programs and attached to the appropriate processing element in the OpenGL pipeline, replacing the fixed functionality of the pipeline.

**GLSL**: a C-like language designed to directly support the development of shaders.