Principles of Computer Graphics - Summary

Andy Tràn

September 27, 2022

This will be a very personalised summary for me to use to study for the course Principles of Computer Graphics (CSCI3260). It might be complete, it might not be, it will probably not be. For questions you can refer to andtran@ethz.ch. This summary is based of the lecture notes and should be used as a supplement to the lectures.

Contents

1 Lecture 1 - 06/09/2022: Introduction, Display and Colour

1.1 Important organizational stuff

- pheng@cse.cuhk.edu.hk, office hours: Thursday 2:30 PM - 4:30 PM, SHB 929
- **Lecture hours:** Tuesday 10:30 am - 12:15 pm, Thursday 11:30 am - 12:15 PM
- **Tutorial hours**: Monday 3:30 pm - 4:15 pm, Thursday 5:30 pm - 6:15 pm
- **Reference book**: Fundamentals of Computer Graphics by Peter Shirley (not necessary), OpenGL Programming Guide
- **Course**: Consists of three parts: introduction, basics in graphics, more about graphics
- **Grading**: (2) programming assignments 0.25, course project 0.20, mid-term exam 0.25, final exam 0.30
- **Release**: Assignment 1 - release 12/9, deadline 02/10, assignment 2 - release 3/10, deadline 30/10, course project - release 31/10, deadline 27/11, mid-term exam 18/10 10:30 - 12:15

1.2 Display and Colour

Lecture Outline

- Display Devices and Basic Terminologies
- Frame Buffer (Memory to Display)
- Color Space: RGB, CMY, HSV, YIQ, CIE YZ
- ALpha Channel and Double Buffering

1.2.1 Display Devices

Mechanism: shoot electrons with varying energy through vertical and horizontal deflectors to hit spot on screen, phosphors on screen jump to excited state when hit by electrons, emit monochromatic light when they drop to rest state
Random scan/Vector scan: give instruction and follow instruction
Raster scan: you go in a line and activate a line, and you turn each line on, where each spot on the screen is called a pixel. You shoot the gun, at the end of the line you turn it off and go back to the start, which is called **retrace**. There is a difference between horizontal and vertical retrace, horizontal is per line, vertical for each following line.
Interlacing: trick to get less flicker out of fixed signal bandwidth, it's like doubling the framerate, for example let's say we have 30Hz, we send two signals but each with a time difference between each other. For example to line 0 we send at 0 and 1, line 1 we send at 2 and 3. Result: doubling the perceived the framerate, without costing more bandwidth
Flat-Panel displays, two classes

- **Nonemissive displays**: LCD (optical effects to split light)
- **Emissive display**: field emission display (FED), light-emitting diode (LED), organic light-emitting diode (OLED). Which is more power efficient

On LCD: we block light instead of emitting the correct light.
FED: thousands of micro-electron guns

1.2.2 3D Displays overview

Two human visual cues used

- Stereopsis: seeing 2 slightly different images in each eye
- Motion Parallax: seeing slighty different images as you move around

Terms used:

- Stereoscopic: difference image to each eye, viewer must wear special glasses
- Autostereoscopic: different image to each eye, does not require special glasses
- Multi-view: different images depending on viewer's position

Note: Multi-view can be combined with used in combination of the others

1.2.3 Stereoscopic and Autostereoscopic displays

- Stereoscopic displays (common): two approaches
  1. using circularly polarized glasses (like in cinemas)
  2. using active shutter glasses (requires batteries, for example 3D TVs)
- Autostereoscopic displays: two approaches
  1. Lenticular lens: bright but blurry, old
  2. Parallax barriers, darker but sharper, like Nintendo 3DS

Downside of Autostereoscopic displays: usually limited to 1 or a very few viewers, and narrow sweet spot for viewing 3D.

1.2.4 Multi-view displays

Usually enabled by tracking the person's head.

1.3 Frame Buffer

Graphical storage (memory) and transformation hardware for digital images. We consider computer images as digital, we want to quantify a space into units (pixels).

1.3.1 Greyscale/Monochrome Frame Buffer

- Intensity of the raster scan beam is modulated according to the contents of a frame buffer
- Each element of the frame buffer is associated with a single pixel on the screen

Each marker corresponds to a pixel on the computer screen, remember rasterizing from the beginning of this lecture
Note: digital to analog converted (DAC)

1.3.2 Resolution

Determined by

- number of scan lines
- number of pixels per scan line
- number of bits per pixel

1.3.3 Colours

- 1 bit: B or W, 8 bit: 0 pure black to 255 pure white, with colours in between. To have true colour we need 8 bit per RGB
- To produce colours we mix intensities of each colour, to have a full spectrum we need a monitor which supports 256 voltages for each colour, the description of each colour in frame buffer memory is called **channel**. The term **truecolour (24 bits)** is for systems which the frame buffer stores the values for each channel (3 channels for RGB)
- **Color table**: for few bits per pixel, we have to map non-displayable colours to displayable ones. We can remap color table entries in software.

1.3.4 Look Up Tables (LUT)

Pseudo color: assign computed values systematically to a gray or color spectrum to indicate differences, for example height, speed, etc.

1.4 Color Space

1. RGB: additive color space, used for displays
2. CMY: subtractive color space, used for printing
3. HSV: (H circular, S distance from axis, V brightness), corresponds to artistic concepts of tint, shade, and tone
4. YIQ: (Y luminance, I orange-cyan hue, Q green-magenta hue), exploits properties of the visual system, used in TV broadcasting
5. XYZ system: defined in terms of three color matching functions

Definition 1 (Gamut) *device's range of reproducible color*

1.5 Alpha Channel and Double Buffering

1.5.1 Alpha Channel

Idea: we store one color per pixel, but we get hard edges. So we introduce an alpha channel next to the RGB channel, to blend with the lower layers to smoothen it out. Can be regarded as **1 - transparency** or **opacity**.

Example 2 (Blending) *We have a source and destination image, we can overlay them and the alpha value denotes of how much percentage we see each image when we overlay them*

1.5.2 Image Matting

What part of the image we want to keep, using a mask.

1.6 Double Buffering

Problem 3 *what happens when we write to the frame buffer while it's being displayed?*

Solution 4 (Double-buffering) *1. Render to the the back buffer and swap when rendering is done*

2. Double the memory

2 Lecture 2 - 08/09/2022: Useful 2D and 3D mathematics

2.1 Coordinate Systems

2.1.1 2D Cartesian Reference Frames

There are two ways of using this system: (a) starting at the lower-left screen corner, (b) starting at the upper-left screen corner.

2.1.2 Polar Coordinates in the XY plane

We start from a center, with a radial distance **r** and the angular displacement θ from the horizontal
We can convert it to the cartesian system: $x = rcos\theta$ and $y = rsin\theta$
To polar system: $r = \sqrt{x^2 + y^2}$ and $\theta = tan^{-1}(\frac{y}{x})$
Definition: $\theta = \frac{s}{r}$, where θ is the angle subtended by the circular arc of length s and r
We know: $P = \frac{2\pi r}{r} = 2\pi$, total distance around P

2.1.3 3D cartesian reference frames

Right-handed system: Take your right hand, palm towards you, thumb (positive x direction) to the right, index finger up (positive y direction), middle finger towards you (positive z direction). Yes, the back of you is positive z.
Left-handed system: Like RHS, but this time away from you is positive z. (Think of how to use your hand to show this)
In OpenGL: right handed (common), DirectX free to choose

2.1.4 Cylindrical-coordinate System

1. The surface of constant r is a vertical cylinder
2. The surface of constant θ is a vertical plane containing the Z-axis
3. The surface of constant z is a horizontal plane parallel to the Cartesian XY plane
4. Transformation from a cylindrical coordinate specification to a cartestian reference system

$X = rcos\theta$, $Y = rsin\theta$, $Z = z$

2.1.5 Spherical-coordinate system

Which is like polar coordinate in 3D space, we have $P(r, \theta, \phi)$
it holds that $x = rcos\theta sin\phi$, $y = rsin\theta sin\phi$, $zrcos\zeta$

Definition 5 (Angles in 3D) *We define it as* $\omega = \frac{A}{r^2}$*, total area is* $\frac{4\pi r^2}{r^2} = 4\pi$

2.2 Points and Vectors

2D vector: $V = P_2 - P_1 = (V_x, V_y)$, length is defined as $\sqrt{V_x^2 + V_y^2}$, angle is $\alpha = tan^{-1}(\frac{V_y}{V_x})$
3D vector: V is same as 2D but with one more value, length is defined equivalently with V_z. We have three direction angles

We have the following rules

1. Addition: $V_1 + V_2 = (V_{1x} + V_{2x}, V_{1y} + V_{2y}, V_{1z} + V_{2z})$
2. Scalar multiplication: $aV = (aV_x, aV_y, aV_z)$
3. Scalar product: $V_1 \cdot V_2 = |V_1||V_2| \cos\theta$, from there you can derive θ
4. Normalization: $\frac{V}{|V|}$, so its own product is 1
5. Perpendicular: $|A||B| \cos\theta = A \cdot B = \begin{cases} 0 & \text{if } \theta = 90deg \\ > 0 & \text{if } \theta < 90deg \\ < 0 & \text{otherwise} \end{cases}$
6. Cross product of two 3D vectors: $V_1 \times V_2 = u|V_1||V_2| \sin\theta$, where u is the unit vector that is perpendicular to both V_1 and V_2
7. Basis vectors: we can specify the coordinate axes in any reference frame with a set of vectors, one for each axis

3 Lecture 3 - 13/09/2022

Continuation of previous lecture Properties of cross product

- Anti-commutative: $V_1 \times V_2 = -(V_2 \times V_1)$
- Non-associative: $V_1 \times (V_2 \times V_3) \neq (V_1 \times V_2) \times V_3$
- Distributive: $V_1 \times (V_2 + V_3) = (V_1 \times V_2) + (V_1 \times V_3)$

3.1 Useful 2D mathematics

1. Distance from P to line AB:
    - Find line direction: $v = (x_2 - x_1, y_2 - y_1) = (d_x, d_y)$
    - Find normal of line by swapping elements in v: $n = (-d_y, d_x) = (y_1 - y_2, x_2 - x_1)$
    - Normalize n and v as $\hat{n}, \hat{v}$
    - Use dot product to find h and l: $h = |(P - A) \cdot \hat{n}$, $l = (P - A) \cdot \hat{v}$
    - Need to check l against length of AB
    - If $l < 0$ or $l > |AB|$, compute point-point distance
2. Line-Line intersection:
    - Express as parametric forms: $AB : (x_1, y_1) + t_{AB}(x_2 - x_1, y_2 - y_1)$, $CD : (x_3, y_3) + t_{AB}(x_4 - x_3, y_4 - y_3)$ and set them both equal
    - If no solution, that meas they are parallel

- Substitute back to the parametric form

3. Which-side test: given a point and a line, Calculate
4. Area of arbitrary polygons: polygon area $\frac{1}{2}\sum det...$
5. Inside-outside test:
   - Method 1: repeat the which-side test for each edge in order (only works for convex polygons)
   - Method 2:Odd-intersection count, side = $\begin{cases} \text{outside}, & number \equiv_2 0 \\ \text{inside}, & \text{otherwise} \end{cases}$
6. Linear Interpolation
7. Barycentric coordinate: by means of area ratios
8. Spherical linear interpolation
9. Normal of a triangle
10. Approximate normal at a vertex (vertex normal vs face normal): average the face normals of neighboring faces

3.2 sth

Properties

- No fixed points under translation: all points move
- Multiple translations are order-independent, since addition is commutative

3.2.1 2D scaling

Properties

- Origin fixed: $x' = 0$ if $x = 0$
- Order independent: $x'' = x' \cdot S'_x = x \cdot S_x \cdot S_x \;= x \cdot S'_x \cdot S_x$

Single out *arbitrary fixed point scaling* at (x_0, y_0) as follows:

$$x' = x \cdot S_X + (1 - S_x)x_0$$
$$y' = y \cdot S_y + (1 - S_y)y_0$$

Rotate by θ:

$$x' = r\cos(\theta + \phi) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\theta + \phi) = r\sin\phi\cos\theta + r\cos\phi\sin\theta$$

Multiple 2D rotations are order-independent Fixed-point: origin dependent

3.2.2 Other 2D transformations

- Reflection (about X/Y - axis): not equal to a rotation, except when reflecting over x and y, then it's a rotation

2D shearing: order dependent

3.3 Matrix operations

3.3.1 Homogenous representations

1. Translation cannot be represented using $2x2$ matrices and homogenous coordinates help
2. We are left-multiplying the matrix on the vector/point

4 REAL Lecture 3: Hierarchical Model

Lecture Outline

1. Graphics Primitives: Points, Lines and Triangles
2. Data structure: vertex list and index list
3. Hierarchical Structure
4. View-world or Modelview transformations
5. Basic scenegraph concept

4.1 Graphics Primitives: Points, Lines and Triangles

- **Idea:** It's all about **coordinates** and **connectivity**
- **Meaning:** wireframe and meshes are built up by points, lines, or triangles. We call this **tesselation/triangulation**
- **Most efficient input type for rendering polygons: Triangle strip** or **triangle fan**. The differences between the two is the following, triangle strip is a set of triangles which share vertices (can share different vertices), while triangle fan has one **(!)** shared center.

4.2 Data structure: vertex list and index list

We have two kinds of data strucutres, namely vertex lists and index lists.

- **Vertex list:** you store all the vertex coordinates in one array
  - Benefit: sequential memory access
  - Negative: very likely to have duplicated vertices in the array, waste of storage
- **Index list/Indexed Face Set:** we have two arrays, one called the **face list**. That one contains all the polygons and the corresponding vertices. While the **vertex list** contains the coordinates for each vertex.
  - Benefit: reuse vertices and keeps a compact vertex list
  - Bad: random memory accesses which lead to cache misses

Note: (1) triangle strip can be implemented on either data structure, (2) in some APIs these data structures are built in, (3) indexing can start with 0 or 1.

4.3 Hierarchical structure

A scene is composed of the multiple objects, which are composed of subjects as well. We can create a hierarchical model and break it down. Each objects has its own polygon list and associated vertex list.

4.4 View-world or Modelview transformations

- **Idea:** we model objects in a model space, which is pretty dumbed down in a cartesian matter where we start from the origin. (**Model space**) This usually doesn't reflect the reality and we want to model it to the common world (**World space**) and then to our eyes perspective (**Eye perspective**).
- **Example:** we create a moon in the object space, we put it in the sky, and then we look at it.
- **Concrete:** An object is defined as $P_{obj} = (x, y, z)^T$ to get it into the world coordinates we need to transform it, namely $M_{obj2world} \times P_obj$ and to get the eye coordinates, $M_{world2eye} \times M_{obj2world} \times P_obj$. **Note:** Since we usually don't care about the steps inbetween, we usually merge $M_{obj2world}$ and $M_{world2eye}$ into one matrix called the **modelview/viewworld** $M_{modelview}$.

Notes about the modelview Matrix $M_{modelview}$:

- OpenGL puts the eye-point at the origin looking towards the negative z-axis
- OpenGL is a state machine: it has an internal memory storage for the modelview matrix
- When calling transformation operations, the kernel constructs a matrix for the transform and right multiplies it with its internal modelview matrix.

4.5 Basic scenegraph concept

- Organize the whole model hiearchy as a tree structure
- Examples of language who do that: VRML, OpenSG

How does this work exactly? We have group nodes, who associate nodes into hierarchies, leaf nodes, who contain all the descriptive data of objects in the virtual world used to render them.

Avoid Modeling Glitches

1. Avoid T-Join
   - Problem: we have edges which sometimes don't touch due to computation and rounding errors of floats
   - Solution: we break a triangle into two triangles.
2. Avoid overlapping polygons in your model
   - Problem: overlapping triangles give flipping colors (**z fighting**)
   - Solution: create another triangle for the overlapping area, or turn off depth test when drawing the next triangle

5 Lecture 4: Interactive 3D Control

Lecture Outline

1. Interactive Control Concept
2. Interactive Translation relative to screen
3. Interactive rotation: rolling ball

5.1 Interactive Control Concept

- **Idea:** we use the mouse as an input device for 3D viewing control
- **Approach:** by modifying the (base) 4x4 Modelview matrix incrementally accordingly to the mouse motion, we can control our 3D viewing interactively.

Normal Modelview Transformation Equation: assume the modelview matrix is affine. In particular, it has translation, rotation and scaling only. If so we can write the Modelview transformation like this:

$$\begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix}.$$

5.2 Interactive Translation relative to screen

Left-Multiply a translation matrix to the existing modelview. We can have a screen-aligned translation, since we don't need to use the fixed point rule for translations.

5.3 Interactive rotation: rolling ball

If we want to rotate our existing modelview, one of the most common mistakes being made is left-multiplying the rotation matrix to our modelview. We need to apply the **fixed point rule**

1. Undo the translation (T_x, T_y, T_z) in the modelview matrix
2. Left-multiply the rotation matrix
3. Redo the translation (T_x, T_y, T_z)

Now we need to construct the rotation matrix, to map our 2D mouse motion to a 3D rotation.

- **Method:** rolling ball
- **Idea:** imagine the object is inside a glass ball of radius r.
- To construct the rotation matrix:
  1. Put the mouse motion in an XY-eye space: (dx, dy, 0)
  2. Axis of rotation perpendicular to the motion vector: (-dy, dx, 0)
  3. Angle of rotation relative to motion vector length: $\sqrt{dx^2 + dy^2}$

6 Lecture 5: Cameras, Projections, and Clipping

Lecture Outline

1. Pin-hole camera
2. Family of Projections
3. Parallel Projection
4. Perspective Projection
5. OpenGL Projection Model: 4x4 projection matrix
6. Clipping and Perspective Division