CS3241 and CS148 Computer Graphics
AY24/25, ST
Notes

Sim Ray En Ryan

December 3, 2025

Contents

1 Blender Basics and Introduction

1.1 Basics

Is a free open source 3D creation suite, supporting:

- Modeling
- Rigging
- Animation
- Simulation
- Rendering
- Compositing
- Motion Tracking
- Video Editing
- Game Creation

For CS148, will learn:

- Modeling
- Lighting
- Shading , Texturing
- Rendering
- Python Scripting API
- Custom raytracer

1.2 Other Applications

Computer graphics also used in:

- Visual Effects (VFX): Photorealistic Simulation and Rendering
- VFX: Creature Modeling, Motion Capture, and Animation
- Video games, AR, and VR
- Scientific Visualization
- Synthetic data generation for AIML
- Anything that has a visual or a GUI

1.3 Relevance to Other Fields

1.3.1 Mathematics

Mathematics used to analyze fluid flows over geometric surfaces, smoothing out high frequencies in noisy functions, and computation for rendering.

1.3.2 Natural Sciences

- Physics: Photonics
- Biology: Motion Capture and Perceptual Color
- Chemistry: Modeling objects and materials

1.3.3 Engineering

High performance computing and aggressive approximations. Also for creating the hardware and systems that will create graphics such as GPUs and cameras.

1.3.4 Arts

The final graphics have to look nice.

2 Geometry and Transformations

2.1 Triangles

Triangles are used as we can optimize the pipeline and hardware for one shape. Triangles can be used to make any other shape, complex shapes are approximated well by triangles, transformations can be applied just on the vertices, and any three points will always be planar

2.1.1 OBJ Files

OBJ files define vertices with v x1 x2 x3, and f v1 v2 v3. For example, a cube will have 8 vertices with each corresponding to the 8 3digit binary numbers, and 12 faces, since 2 triangles make a square, and there are 6 squares for a cube.

2.1.2 Subdivision

Subdivide triangles into 4 triangles, and then move the old and new vertices. This results in C^2 continuity for most cases other than C^1 at extraordinary vertices.

TODO: Insert formula for pertubing here.

2.1.3 Interpolation

When moving one vertex, the others will be interpolated and move smoothly along. Linear splines are not smooth enough, while cubic spliens are usually enough to have C^1 continuity everywhere.

2.1.4 Cardinal Cubic Splines

Interpolate the second and third points.

2.2 Transformations

2.2.1 Local vs Global

Each object has a local coordinate system (object space), and it can be then placed into the scene (world space). Then, rotating, scaling, and translating in the world space will move the object anywhere we want.

2.2.2 Rotation

Each axis has their own Rotation Matrix, based on theta as well. Rotation preserves shape, and is non-commutative (since matrix multiplication is not commutative)

2.2.3 Scaling

The matrix is a diagonal matrix.

2.2.4 Translation

USe the homogenous coordinates, and the translation matrix is simply the I3 matrix, with the translation vector x y z to its right, a zero vector below, and a 1 at the last index. To make things consistent, rotation and scaling are also expressed in homogenous coordinates. To convert them back to a 3D shape, simply remove the fourth component.

3 Rasterization and Shading

3.1 Rasterization

An image is a 2D grid of colors, and may have multiple channels. In the RGB color space, each pixel can be represented by $[(0,0,0),(1,1,1)]$. Converting a shape from lines onto pixels is known as rasterization.

One such method is if the center of the pixel is in the triangle, then color it with the triangles color.

3.1.1 Determining if pixel should be colored

Each triangle is projected onto the 2D Grid, and is made up of three lines.

Let the points on the triangle be p1, p2, and p3, defined counter-clockwise. Then, find $(p - p1) \times (p2 - p1)$ where p is the point in question. If the cross product is negative, it is 'inside' the triangle. If a point is in all three half-planes, it is inside the triangle $(max(S1(p), S2(p), S3(p)) < 0)$

Not a perfect solution, as it may not be very efficient the more triangles there are, many triangles can intersect at a pixel center. Another approach is to color a pixel based on how much of the triangle is within the pixel.

3.2 Shading

3.2.1 Flat Shading

The fastest, which is the color every pixel inside the triangle with the average color of the vertex. It does not look good and by 'hardcoding' the color in the pixel, it is not robust enough to reflect changes in light

3.2.2 Phong Reflection Model

Each vertex now stores a normal vector, and the color of each pixel is now determined by:

- Ambient: How an object looks in the dark, a constant RGB value $c_{ambient} = c_a$
- Diffuse: How objects look when light bounces off of it. $c_{diffuse} = c_d c_l n \cdot l$, where Cd is the diffuse material of the surface, and Cl is the color of the light. Then, take the dot product of the normal vector and the vector to the light to obtain the strength of the light on the object. Note that both vectors must originate from the same point of the surface. Usually encapsulate with $max(0, n)$ to make sure objects facing away from light do not have additional computations.
- Specular: How shiny a material is. $c_{specular} = c_s c_l max(0, e \cdot r)$ where Cs is the specularity of the surface, r is the reflection vector, and e is the vector to the camera.

3.2.3 Specularity

Additionally, specularity has a shininess value called the Phone exponent that allows the tuning of the shinyness of the material. It is the exponent of the $max(0, e \cdot r)^a$ component.A higher a value will make that component smaller and thus less shiny. h is the vector planar with l and e that bisects. The angle between n and h can be used to approximate the angle between l and e.

Furthermore, it is easier to approximate $c_{specular}$ with $c_s c_l max(0, n \cdot h)^a$, where $h = \frac{e+l}{|c+l|}$, and normal is the normal vector that is already calculated.

3.3 Smooth Shading

Flat shading is simple, renders fast, but looks bad. Gourad Shading and Phong Shading are barycentric interpolations. Gourad Shading has a good balance between speed and result, while Phong looks the best but takes the longest.

3.3.1 Gourad Shading

For each point v inside the triangle, use it to subdivide the triangle into three more triangles. Then, the color $c_v = \frac{A_a}{A_{total}} c_a + \ldots$ for all three sides.

3.3.2 Phong Shading

Instead of using the color of the vertex, calculate a n_v with the formula above to obtain a normal for each point. Then, use the Phong Reflection Model to calculate the color at each point.

3.4 OpenGL Pipeline

- Vertices
- Transformed Vertices: Translate, Rotate, or Scale based on position
- Primitives: Getting the faces and the normals for each triangle for each pixel
- Shaded Fragments: Compute the color in each pixel in a triangle

4 Colour, Images, and Cameras

4.1 Light

Information can be determined from incoming, reflected, and specular energy, We simulate this with computer graphics.

4.1.1 Human Eye

Has 3 color cone cells, and 1 rod cell. Rod is very sensitive to low light levels. During day, rod signals are tuned out, and cone cells are used to perceive color (Red, Green, Blue wavelengths) in their intensity and color.

Thus, computer screen use RGB as that is what our eyes are sensitive to, and what is most similar to our vision. In the real world, a yellow object absorbs all wavelengths except for yellow, and our 3 rods perceive different signals. The computer screen can simulate this with RGB wavelengths, and humans will perceive them the exact same way.

The pupil (lens) prevents all light from getting in, giving us depth. Pupils dilate to allow more light to come in.

There are other models, such as CMYK, YUV. For color intensity, we use 256 (because it is one byte, but also because "the perceptual difference becomes too small"). HDR can use an even larger range.

4.1.2 Screens

Each display has a different output (gamut), and the same digital color can be shown different on each screen. The gamut shows how red, green, and blue a screen can actually go.

4.2 Projection

4.2.1 Coordinate Frames

Local Space goes to World Space goes to View Soace goes to Clip Scpae goes to Screen Space. These are done via matrix multiplications.

$p_{img} = M_{obj2img} \cdot p_{obJ} = M_{cam2img} M_{world2cam} M_{obj2world} \cdot p_{obj}$

4.2.2 Cameras

We use a pinhole camera. In the real world, this requires a long amount of exposure and has limits to the aperture size.

In the virtual world, there is no such problem, and the pinhole camera approach will still allow objects:

- To be blocked (occluded) by other objects

- To make objects further away smaller due to the smaller angular difference

Since images get flipped in the real world (that our brains flip back automatically), we place the film plane at the same distance away to obtain the image.

With the pinhole at origin (0,0,0), the points (x,y,z) get projected to (x',y',h) or $(\frac{h}{z}x, \frac{h}{z}y, h)$ which allows depth to be perceived.

4.2.3 Coordinates

Homogenous coordinates are used again to allow for non linear projections. Add a 1 to the end of the XYZ vector, and left multiply it by diag(h, h, h) with a 1 below the final h.

4.2.4 Viewing Frustrum

Only render objects that project to within the boundaries of the film, and only render objects further away from the camera than the film (front clipping plane). A back clipping plane is also added, and the volume (3d Trapezoid) is the viewing frustum.

Then, instead of trying to project all points onto a fixed (near) plane, the frustum is projected to a normalized cube. The cube is then bounded, and then projected onto a screen.

4.2.5 Z-Buffer Algorithm

Then, use x and y for the image space position, and z to determine the depth. The currentdepth is set to be negative infinity. For each cooridnate xyz and color c, if the depth (z-buffer) of xy is less than the current depth, render the color of that pixel to c, and then update the depth to the new depth. This is to ensure that the closest pixel is rendered.

4.3 Lens Based Cameras

Unlike pinholes, lenses require a shorter exposure, and will make it such that only some objects will be in focus. It mimics the anatomy of the eye more closely.

4.3.1 Focus distance

Focal distance refers to how far an object has to be to be in focus. It is determined by the focal length and the distance of the lens to the sensor.

4.3.2 Focal Length

Depends on each lens. A weaker lens focuses light less intensely.

4.3.3 Field of View

Depends on sensor size and focal length. Wide angle lens have wider field of view due to a shorter focal length.

4.3.4 Depth of field

Blurring of objects no in focus (circle of confusion). In games, motion blue and depth of field are used to make things 'imperfect' and more realistic. The blurring also allows a hack in which further objects can be blurred and not need to be rendered.

5 Light and Optics

5.1 Modelling Light

5.1.1 Steradians and Intensity

A cone like volume defined by a point and a surface area patch. Just as how there are 2pi radians in a circle, there are 4pi steradians in a sphere. The radiant intensity of a light source per steradian is $I(w) = \frac{dP}{dw}$. For isotropic point lights (uniform), integrating it gives $dP = Idw$, $P = \int_{sphere} Idw = 4\pi I$. For anisotropic light sources, since intensity varies, a function of steradians is needed.

Irradiance $E = \frac{dP}{dA}$ is the power per unit surface area of the object that the light is hitting. When tilting the surface, $E_{tilt} = Ecos\theta$, since the light angles are coming at an angle and are not as direct (Think seasons). The irradiance also varies based on the distance from the light source.

5.1.2 Radiance

$L = \frac{dI}{dAcos\theta}$, since in the real world, light does not come from a point. Approximate the area light comes from by breaking it into smaller areas, eventually as points.

5.1.3 Color Bleeding

For example, in mirrors (what you see in the mirror 'acts' as a light source), or simply normal objects (such as holding a red paper close to a white object). You can measure incoming light using a chrome sphere, or light probe. With this object, we can then overlay it onto a CGI object to make it look as though it has all the color bleeding it should actually have.

5.2 Modeling Materials

Light interacts with an object based on the material of the object. It can be absorbed, reflected, or transmitted (scattered).

- Bidirectional Reflectance Distribution Function: When light rays hit an opaque object, it gets reflected at that point
- B Transmittance DF
- B Surface Scattering RDF: Some light is reflected, some gets absorbed and transmitted elsewhere. For example, skin, or marble should be slightly transluscent.

An object material data is measured using a gonioreflectometer, or with simple analytical models:

- Blinn-Phong: Plastic

- Cook Torrance: Specular Metal
- War: Anisotropic: Brushed Metal or Hair
- Oren-Nayar: non-matte

Objects can usually be simulated by one of the models. For example, water and glass are both approximated well by a transparency shader.

5.2.1 BRDF

$BRDF(w_i, w_0) = \frac{dL_0(w_0)}{dE_i(w_i)}$

The ratio of the incoming light to the outcoming light: irradiance to radiance. BRDF comes with 3 for each color component.

5.2.2 Light Equatiom

Given a point on an object, we have

- Light from an incoming direction w_i
- A corresponding outgoing direction(s) w_0
- A BRDF model which says how much light gets reflected in each w_0 given w_i

  In total, $L_o(w_o) = \int_{i \in in} BRDF(w_i, w_o) L_i cos\theta_i dw_i$, where the BRDF function models the material of the object, is scaled by the incoming radiance, and then scaled by the cosine term due to the titlt.

6 Raytracing I

6.1 Ray in Computers

Rays can be represented as an equation for derivation purposes. For calculation, since we only need to know the effect of the ray, we can define it with

- Origin Point
- Vector Direction
- Parameter t for time

At each pixel, shoot a ray $R(t) = A + (P - A)t$, where A is the camera position, P is the center of the pixel. $t \in [1, t_{far}]$ of the frustum. Upon finding the smallest value of t, use that material there to calculate the color of the pixel using the light equation from above.

6.1.1 Ray Triangle Intersection

Method 1 Since triangles are planar, first check if the ray intersects the plane within the frustum. A point P is on the plane if the vector between p and a triangle vertex dot product the normal is 0. Subbing into the equation, we get $t = \frac{(p_0 - A) \cdot N}{(P - A) \cdot N}$.If it does, then check if it intersects within the triangle. For example, project both the intersection point and triangle into 2D. Then, use techniques from 2D rasterization. Alternatively, use dot product again with three triangle sides to check if it is interior to the triangle (less than 0)

Direct Intersection Any point in the triangle is a sum of one of the vertices and scaling of two edge vector $p = p_0 + \beta_1 u + \beta_2 v$. In the end, can solve matrix equation $(u, v, A - P) \cdot (\beta_1, \beta_2, t)^T = A - p_0$, where $\beta, \leq 1, t \in [1, t_{far}]$.

6.1.2 Ray Sphere Intersections

A sphere can be represented by an equation. We can substitute it in as well to check for solutions of a quadratic equation which will determine if there's 2, 1, or 0 solutions,

6.1.3 Ray Superquadric Intersections

Similar to the equation of the circle, any superdquadric can be represented by $|x|^r + |y|^s + |z|^t = 1$. The same method can be used to determine intersection

6.2 Computation

Ray tracing was too slow for real time rendering. Optimization was made in both hardware (GPUs) and software (shared memory, spatial locality) to parallelize the computation.

Objects are also approximated as bounding volumes or rectangles to cut away much computations if the ray does not even come close to the object.

6.2.1 Bounding VOlume Hierarchy

By creating a tree of volumes that split into smaller volumes, an O(n) operations speeds up to O(log n). An octree was initially made for this problem, starting with a cube surrounding the whole objects and splitting into 8 smaller cubes in each iteration if needed. BVH is still expensive if there are many objects in the scene (such as a flock of birds). In that case special partitions such as a uniform grid might work better.

6.3 Constructing Rays

After finding the intersection, do lighting to determine the color of the pixel. The light equation has a normal component, and normals are usually defined in object space. Lighting computations are done in world space, and transformations on the normal may not preserve the normality (perpendicular) of the vector. Instead, multiply it by the inverse of the transpose.

6.3.1 Shadow Rays

For each light, cast a shadow ray in the direction of the light. The origin will be the intersection point, and the direction is towards the light.

Due to computer computation, some shadows may start under the surface, resulting in wrong lighting (spurious self occlusion). A simple way to fix this is to perturb (move) the origin slightly in the direction of the normal. The light direction can then also be fixed to make it look more accurate.

7 Raytracing II

Reflection and Transmission usually result in a lot of light bouncing around, and a depth field is used to prevent infinite recursion and stack overflow.

7.1 Reflections

Light is reflected when it hits a surface. Using the dot product of the Normal and the Incident Ray, multiply it by two and project it onto the normal ray, we can obtain a point where the reflected ray should go.

Due to spurious self intersection, perturb it a bit.

7.2 Transmission

Using Snell's law, can service a giant equation for how light slows down and bends in different medium, based on the angle of incident and the IOR (index of refraction) ratio.

When exiting the object, the ratio inverts. A check "isInsideObject" may be used to determine this.

7.2.1 Total Internal Reflection

When light moves from a denser medium to a less dense one, if the bend is enough, total internal reflection will occur instead. Using the giant formula, this happens at any angle higher than the critical angle, and by the giant formula it happens when the square root term is negative.

In graphics, objects are often both reflective and transmissive. When looking at something from overhead, refraction and transmission decrease. When you view things form parallel, you may see better reflections and stronger refractions.

The amount of reflection is based on both the Fresnel equations as well as the angle at which it hits, due to light polarization.

7.3 Attenuation

When light is transmitted through objects, some light is absorbed and scattered. For example, shallow water is clear but deeper water attenuates red light and looks darker, then green light, then blue light as well.

Attenuation along a ray can be described by Beer's law

7.3.1 Caustics

Caused by refraction, when imperfections in the surface of the material to direct lights in certain ways (think funny light patterns in water)

8 Sampling and Texturing

8.1 Sampling

When using area lights, we simulate it as a bunch of lights. Increased the area of the area light will require much (square) more point lights, causing computation to go much slower. Instead, we random sample the rays on the light surface to simulate the effect while reducing computation time.

Area lights result in softer shadows

8.1.1 Anti Aliasing

Sending one ray through each pixel makes edges pixelated. Instead, sample camera rays through random points of the pixel, and then averaging the render results causes a smoother image even though it is the same amount of pixels.

8.1.2 Depth of Field

Random sample camera rays and average the result.

8.1.3 Color Bleeding

For each object, create a new scatter ray starting from the intersection point and going out in a randomly sample direction. Recurse until it hits a light, and anything it hits on the way will be dyed a bit by this color.

8.2 Texturing

Objects are made of polygons. We can apply a texture map (giftwrap) on top of the polygon map (UV map) so images can go onto geometry. Formally, UV coordinates are assigned to each surface, and a lookup table of images is then transformed onto the object.

Barycentric interpolation is used to make distorted objects look more correct when they are views from an angle

8.2.1 UV Unwrapping

Can create seams, and then project, flatten, or warp the object.

8.2.2 Other maps

- Normal maps does not change geometry but can be used to simulate light bouncing off of an object and height even though the object is flat.
- Bump mapping: Uses a 1D height map which is grescale
- Displacement mapping: Uses on the fly subdivision and modifies geometry

8.2.3 Environment Maps

Obtain realistic global illumination by placing scene inside a textured cube or sphere and treating that as a light source.

9 Simulation and Animation I

9.1

10 Simulation and Animation II

10.1

11 Advanced Topics

11.1

12 Art of Images

12.1

13 Next Steps in Graphics

13.1