Raytracing and Shadows

Dr. Sheng Bin(盛斌)
Shanghai Jiao Tong University
Lecture 8

Lecture Overview

- Ray Tracing
  - Ray Casting
  - Light Reflection & Light Rays
  - Object/Ray Intersections
  - Acceleration
- Shadows
  - Hard &Soft Shadows
  - Shadow Mapping
  - percentage-closer soft shadows
  - Irregular Z buffer Shadows
- Official GeForce RTX Real-Time Ray Tracing Demo

What is ray tracing?

- A technique capable producing a very high degree of visual realism
- A technique capable simulating a wide variety of optical effects, such as reflection and refraction, scattering, and dispersion phenomena.
- Best suited for applications where taking a relatively long time to render a frame can be tolerated, such as in still images and film and television visual effects, and more poorly suited for real-time applications such as video games where speed is critical
- It works by tracing a path from an imaginary eye through each pixel in a virtual screen, and calculating the color of the object visible through it.

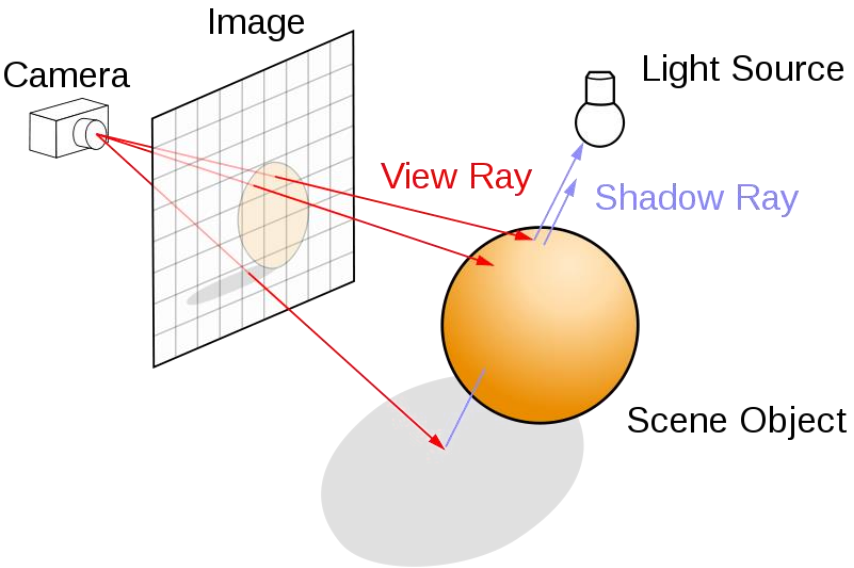
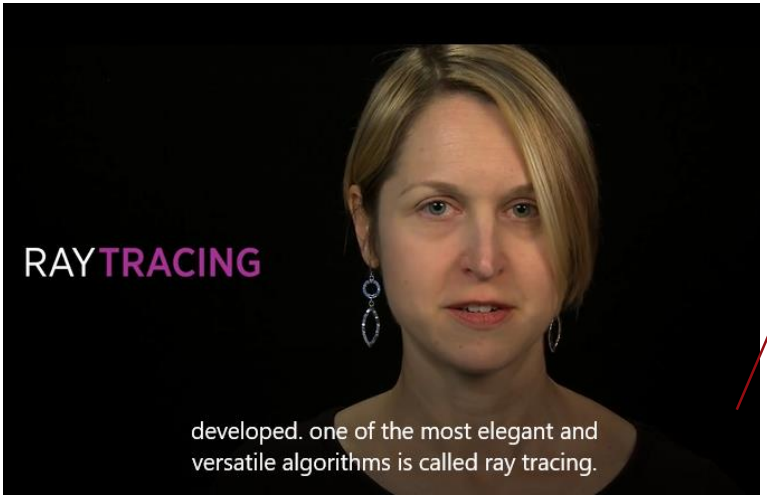


Image
Camera
Light Source
View Ray
Shadow Ray
Scene Object

Outline in Code

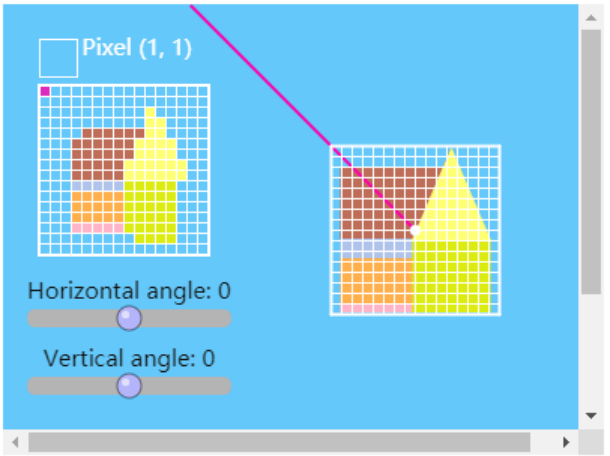
```
Image Raytrace(Camera cam, Scene scene, int width,int height)
{
Image image=New Image(width, height);
for (int i=0; i<height; i++)
    for (int j=0; j<width; j++)
    {
        Ray ray=RayThruPixel(cam,i,j);
        Intersection hit=Intersect (ray,scene);
        Image[i][j]=FindColor(hit);
     }
 return image;
}
```



Play the video "What is ray tracing"

- Online practice:
- https://www.khanacademy.org/partner-content/pixar/rendering/rendering1/e/ray-tracing

- The exercise online is really amazing!!!
- Please try !!!!

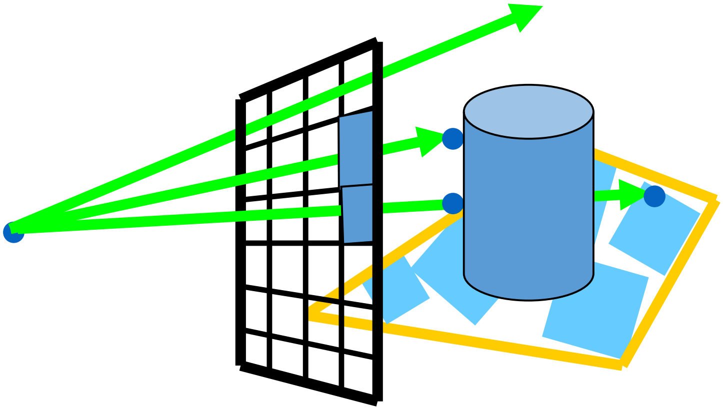


Ray Tracing vs Rasterization

- Raytracing first goes to each pixel, and figures out what it should do at each pixel and then go over each object.
- It has historically been slow because the cost is the number of pixels times the number of objects.
- There are things like acceleration structure now, which is good for walkthroughs of extremely large models (amortize preprocessing, low complexity)

- In contrast, rasterization first goes to all of the objects in the scene. And then, for each of the object, to see which pixels are needed to consider.
- Still pay the cost of the number of the objects, but the number of pixels is just the number of the pixels a given object covers, which could be 10 or 20.

Ray Casting



Confused among ray casting ,marching & tracing?

- Ray casting, ray tracing and ray marching all tend to be used interchangeably as a generalists catch-all term for a rendering process which relies on computed lines of intersection, but there are subtle distinctions implied when speaking precisely.

- Ray casting is a process where points of intersection with a line are computed analytically, using formulas of intersection.
- For instance, when I'm trying to figure out where a user's touch on a touch screen device (screen space) is in world space, I cast a ray from the camera and find out where it intersects the game world. I might also cast a ray down from a character to figure out if they are above the ground, or forward to see if they are about to collide with something.

- Ray marching is a specific algorithm, a variant on ray casting where samples are taken down a line to test for intersections or other criteria. This is easier to implement and allows for speed optimizations via number of samples, but is not as precise even when large numbers of samples are used.

- Ray tracing is a more complex series of tasks to render a 3d sence, which uses ray casting and/or ray marching to compute not only the point of intersection between origin and object surface (or voxel cell etc) but which iteratively computes secondary and tertiary rays, which can be used to collect data used typically (but not exclusively) for calculation of reflected or refracted light.
- As a rendering technique, ray tracing is generally too slow to be effectively used in game development.

Light reflection

Play the video: Light reflection

Light rays

Play the video: Light rays

Question &Thinking

- A surface will be display specular highlights if it is
  - ✓ choose 1 answer
  - Transparent
  - ✓ Shiny
  - Fuzzy
  - Colorful
- ( ) is caused by light reflecting at a random directions on a surface.
- ( ) is caused by light reflecting in a single direction on a surface.

  - ✓ Diffuse light
  - ✓ Specular light

Question &Thinking

- In the image below we can see the tennis ball on the surface of the pool ball.
- If we want to simulate this effect in a ray tracer, what rays do we use?

✓ choose 1 answer
- Camera rays
- Specular rays
✓ Reflected rays
- Refracted rays

Question &Thinking

- How many refraction rays are shown in this image?
- And how many reflection rays?

Hints

- The primary ray travels from the camera and hits the surface of the car.
- The car is shiny, so **a reflection ray** leave the car surface, travelling towards the window.


- Two rays leave the window:
  - 1. **a reflection ray** goes left. 2. **a refraction ray** goes through the window to the right.

The reflected ray hits the surface of the pink car sending another **reflected ray** towards the window.

- Finally ser left.


Question &Thinking

- If we created the scene below in a ray tracer, there would be a shadow ray going from the ( ) to ( ).

- ✓ Light
- Camera
- ✓ Cross on the tennis ball
- A random direction

Question &Thinking

- In the image below we can't really see the entire ball:

- With only one flashlight is it possible to illuminate all sides of the ball?

- choose 1 answer
- No you' d need another light.
- Yes, by holding the light very far away
- ✓ Yes, by bouncing light off another object, such as a white card.
- It depends on the type of ball surface.

Question &Thinking

- The face behind the glass appears warped. If we want our ray tracer to simulate this effect we need to calculate the direction of the ( ) rays.

- choose 1 answer
- Primary
- Shadow
- Reflected
- ✓ refracted

Question &Thinking

Which of the following object would result in refraction rays?

- choose all answers that apply
- ✓ A solid glass statue
- ✓ A glass of water
- ✓ An empty glass
- A polished metal cup

Hints

- Refraction occurs when the light travels from one transparent material to another transparent material with a different density.
- Glass and water are transparent, but metal is not.
- So refraction rays are created for:

  - ✓ A solid glass statue
  - ✓ A glass of water
  - ✓ An empty glass

Question &Thinking

- What causes the red highlight to appear on her cheek?

- choose 1 answer
- Reflacted rays
- Color rays
- Shadow rays

✓ Reflected rays

Interesting results in rendering

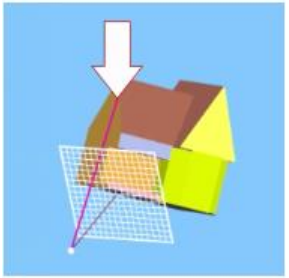
Play the video: Rendering Mike Wazowski

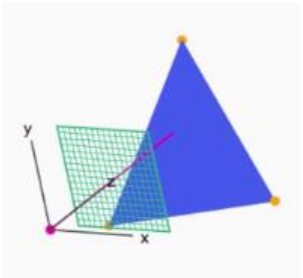
Ready to dive into math?

Ray object Intersection

Ray object Intersection

- Ray object intersection is the most fundamental calculation a ray tracer performs.





- Since objects in our scenes are modeled using millions of tiny triangles, each intersection is between **a ray and a triangle.**

Ray object Intersection

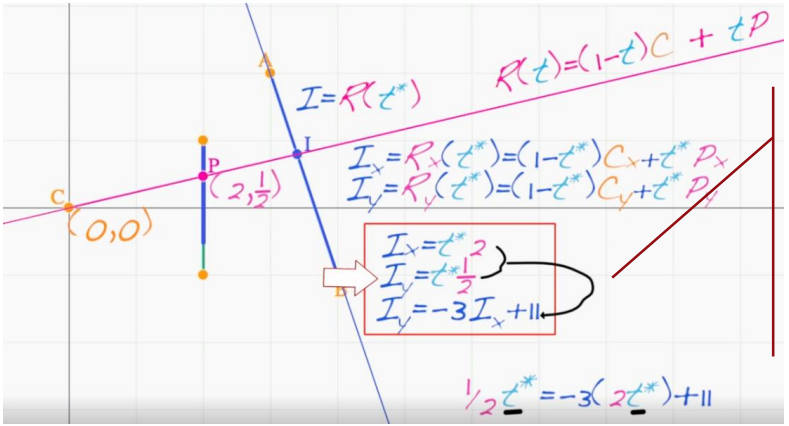
- We'll start with the a simpler version of this problem, the intersection of a ray and a line in 2D.

$$aI_x + bI_y + cI_z + d = 0$$

$$I_x = (1-t^*)C_x + t^* P_x$$
$$I_y = (1-t^*)C_y + t^* P_y$$
$$I_z = (1-t^*)C_z + t^* P_z$$

- Finally we'll extend these ideas from two dimensions to work in three dimensions. By the end of this lesson we'll need to solve a pretty meaty system of equations with 4 unknowns.

Calculate intersection point

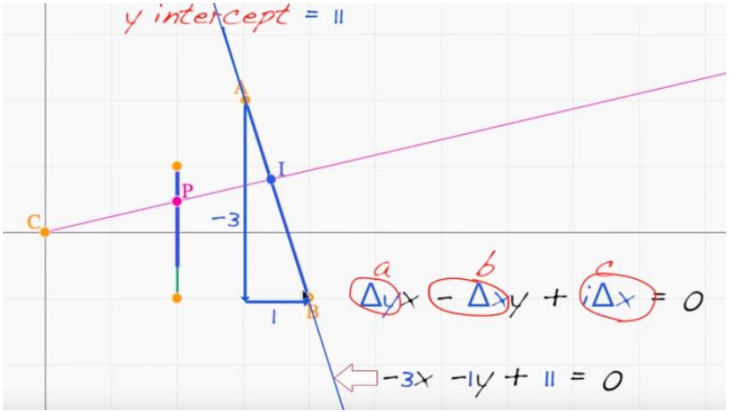


Play the video: Calculate intersection point

Find the code and implement here!

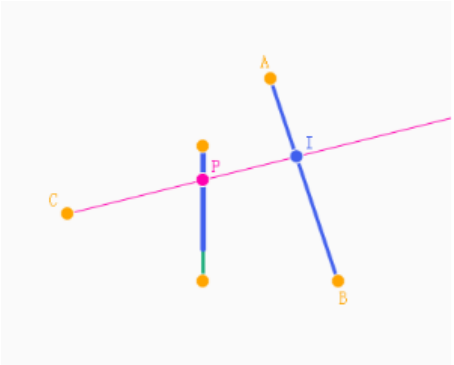
https://www.khanacademy.org/pixar/ray-tracing-in-2d-v2/5119189004845056

Line equation (Implict form)



Question &Thinking

- In the diagram below, what does I, C,P represent?
- With what will every ray generated by ray tracing algorithm touch?

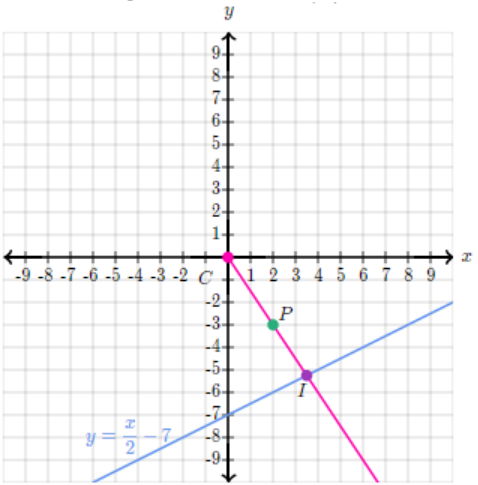


✓ choose the answers

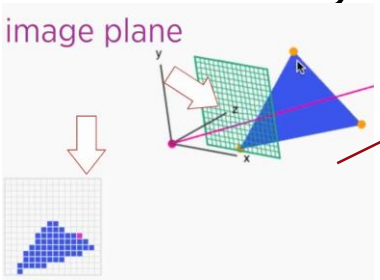
- The ray tracer.
- ✓ An intersection point on the object.
- ✓ The camera.
- ✓ A pixel on image plane.

Question &Thinking

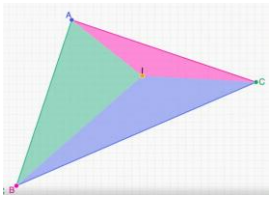
- We can parameterize the ray from C through P as a function of t:
- *R(t)=(1-t)C+tP*
- With C at (0,0) and P at (2,-3), R(t) intersects a line defined by the equation:
- $y = \frac{x}{2} - 7$
- If the intersection point is I and I=R(t^*), what is the value of t^* ?

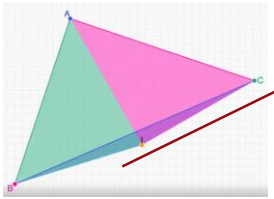


3D Ray object Intersection



Play the video:
3D ray tracing part1





Play the video:
3D ray tracing part2

to determine I is inside or outside the triangle

Question &Thinking

- We are using ray-tracing to create an image of image of a plane that's defined by the equation, $3x + 3y + 5z - 13 = 0$
- We draw a ray from the camera at coordinate (0,0,0) through a pixel at coordinate (5,2,1), and find that it intersects with the plane at point I.
- What are the coordinates of I ?

Question &Thinking

- Triangle ABC is defined by 3 points:
- A=(-2,2,2),B=(3,-3,2),C=(4,1,-4)
- A ray intersects the plane defined by ABC at a point, I.
- If I=(1.4, 0.6, -1.2), is I inside triangle ABC?

Acceleration

Acceleration

Testing each object for each ray is slow

- Fewer Rays
  - Adaptive sampling, depth control
- Generalized Rays
  - Beam tracing, cone tracing, pencil tracing etc.
- Faster Intersection
  - Optimized Ray-Object Intersections
  - Fewer Intersections

Acceleration Structures

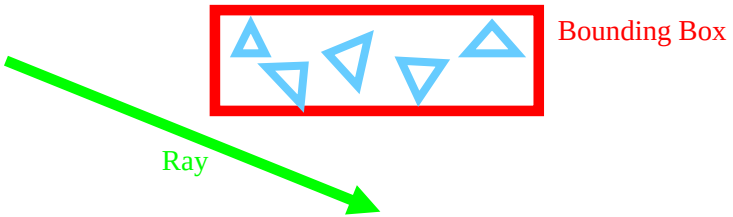
- When given a ray, we want to ensure that test as few objects as possible: Ideally only the object the ray will hit.

- This is not possible but we can consider the number of intersections to be logarithmic in the total size of the number of the objects.

Acceleration Structures

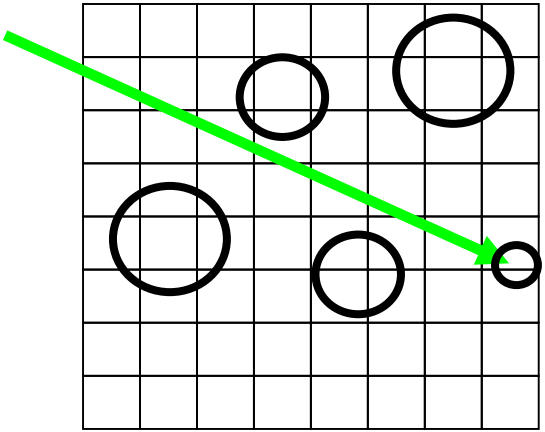
Bounding boxes (possibly hierarchical)

- If no intersection bounding box, needn't check objects



Spatial Hierarchies (Oct-trees, kd trees, BSP trees)

Acceleration Structures: Grids



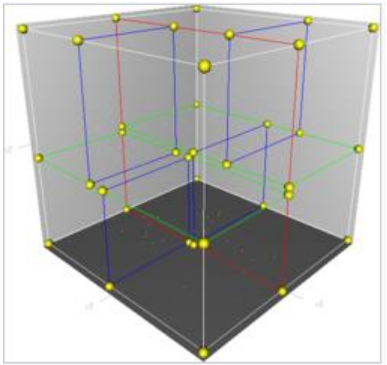
- Regular grids are the simplest acceleration structure.

Acceleration and Regular Grids

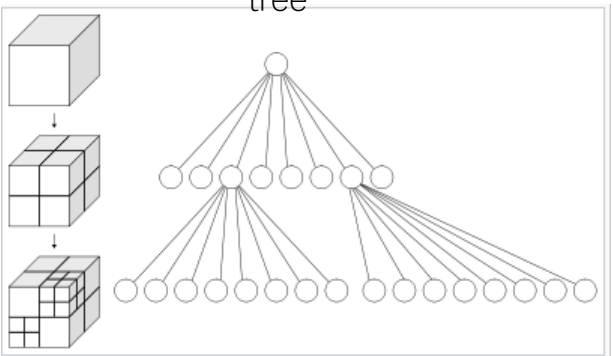
- Simplest acceleration, for example 5*5*5 grid
- For each grid cell, store overlapping trangles
- March ray along grid (need to be careful with this), test againist each triangle in grid cell
- More sophisticated: kd-tree,oct-tree, bsp-tree
- Or use (hierarchical) bounding boxes.

Acceleration and Regular Grids

kd-tree



oct-tree



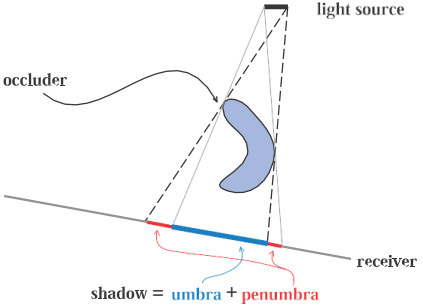
Left: Recursive subdivision of a cube into octants. Right: The corresponding octree.

Shadows

Shadows

- Important for creating realistic images and in providing the user with visual cues about object placement.

  - Occluders: objects that cast shadows onto receivers
  - Umbra: a fully shadowed region of each shadow
  - Penunmbra: a partially shadowed region



Shadow terminology: light source, occluder, receiver, shadow, umbra, and penumbra.

Hard & Soft Shadows

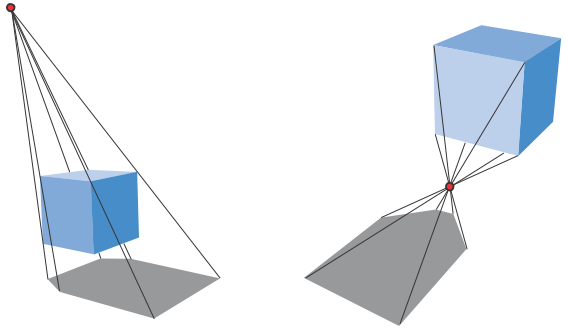
Hard shadows:

- fully shadowed regions generated by punctual light sources with no area
- faster to render than soft shadows
- look less realistic and sometimes be misinterpreted as actual geometric features

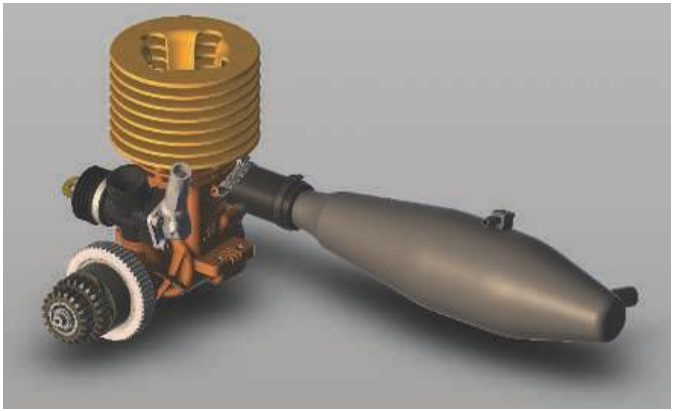
Soft shadows:

- are produced if area or volume light sources are used.
- are recognized by their fuzzy shadow edges.
- generally cannot be rendered correctly by just blurring the edges of a hard shadow with a low-pass filter.
- The closer the shadow-casting geometry is to the receiver, the sharper a correct soft shadow is.
- The umbra region of a soft shadow decreases in size as the light source grows larger, and it might even disappear, given a large enough light source and a receiver far enough from the occlude.

A mix of hard and soft shadows. Shadows from the crate are sharp, as the occluder is near the receiver. The person's shadow is sharp at the point of contact, softening as the distance to the occluder increases. The distant tree branches give soft.



At the left, a correct shadow is shown, while in the figure on the right, an antishadow appears, since the light source is below the topmost vertex of the object.



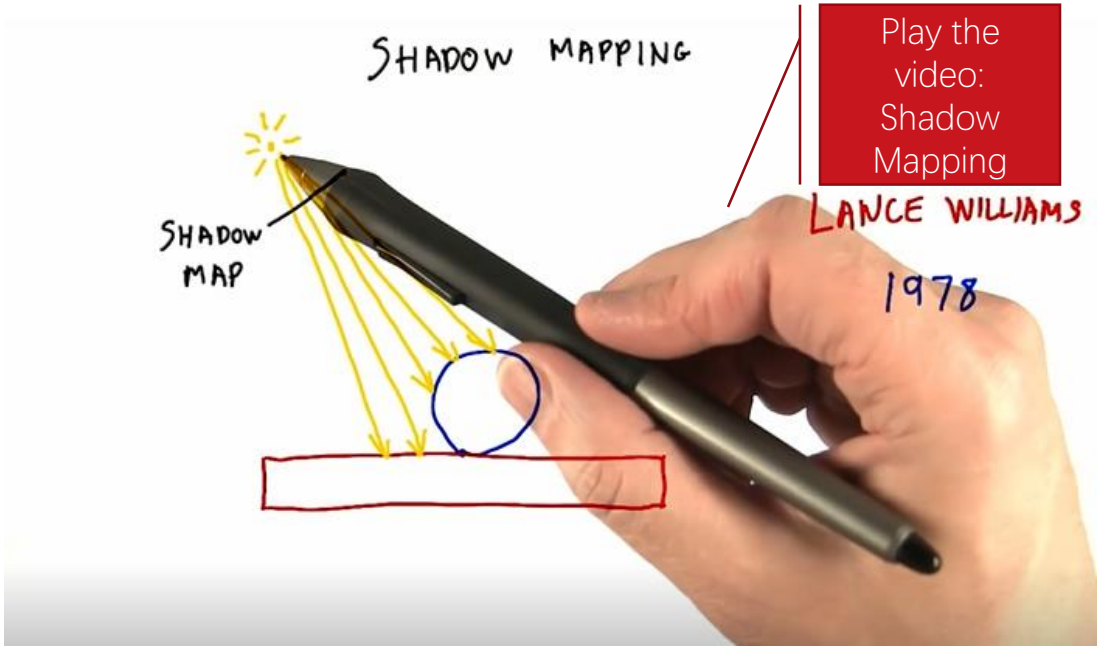
Drop shadow. A shadow texture is generated by rendering the shadow casters from above and then blurring the image and rendering it on the ground plane.

Shadows Maps

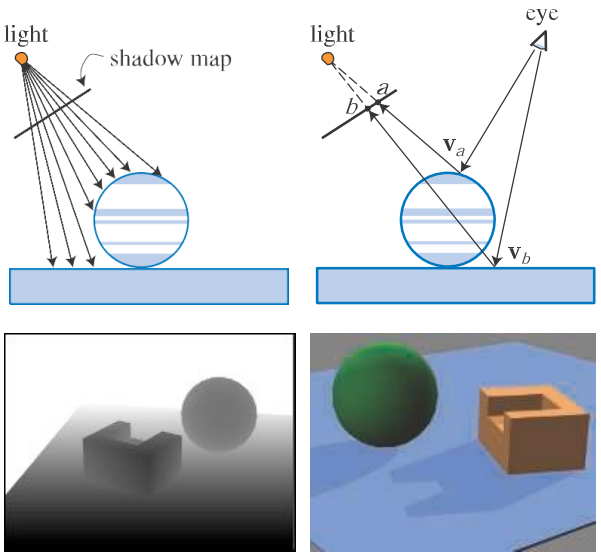
- A common z-buffer-based renderer could be used to generate shadows quickly on arbitrary.
- The idea is to render the scene, using the z-buffer, from the position of the light source that is to cast shadows.
- When this image is generated, only z-buffering is required. Lighting, texturing, and writing values into the color buffer can be turned off.
- Each pixel in the z-buffer now contains the z-depth of the object closest to the light source, and the entire contents of the z-buffer is the shadow map.

Shadows Maps

- Shadow mapping is a popular algorithm because it is relatively predictable.
- The cost of building the shadow map is roughly linear with the number of rendered primitives, and access time is constant.
- The shadow map can be generated once and reused each frame for scenes where the light and objects are not moving, such as for computer-aided design.



Play the video: Shadow Mapping



- Shadow mapping.
- On the top left, a shadow map is formed by storing the depths to the surfaces in view.
- On the top right, the eye is shown looking at two locations. The sphere is seen at point $\mathbf{v_a}$, and this point is found to be located at texel a on the shadow map. The depth stored there is not (much) less than point va is from the light, so the point is illuminated. The rectangle hit at point $\mathbf{v_b}$ is (much) farther away from the light than the depth stored at texel b, and so is in shadow.
- On the bottom left is the view of a scene from the light's perspective, with white being farther away.
- On the bottom right is the scene rendered with this shadow map.

- One disadvantage ：the quality of the shadows depends on the resolution (in pixels) of the shadow map and on the numerical precision of the z-buffer.
- Since the shadow map is sampled during the depth comparison, the algorithm is susceptible to aliasing problems.
- A common problem is self-shadow aliasing, often called “surface acne” or “shadow acne,” in which a triangle is incorrectly considered to shadow itself.
- This problem has two sources:
  - the numerical limits of precision of the processor
  - geometric (samples generated for the light are almost never at the same locations as the screen samples) .
- When the light’s stored depth value is compared to the viewed surface’s depth, the light’s value may be slightly lower than the surface’s, resulting in self-shadowing.
- Introducing a bias factor can help avoid various shadow-map artifacts.

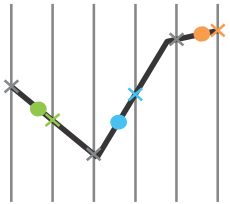
Shadow-mapping bias artifacts.

On the left, the bias is too low, so self-shadowing occurs.

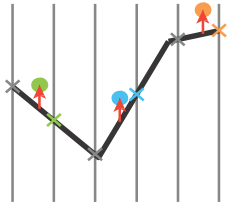
On the right, a high bias causes the shoes to not cast contact shadows. The shadow-map resolution is also too low, giving the shadow a blocky appearance.

Shadow bias

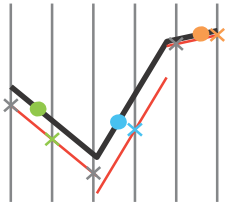
- The surfaces are rendered into a shadow map for an overhead light, with the vertical lines representing shadow-map pixel centers. Occluder depths are recorded at the × locations.
- We want to know **if the surface is lit at the three samples shown as dots**. The closest shadow-map depth value for each is shown with the same color ×.



If no bias is added, the blue and orange samples will be incorrectly determined to be in shadow, since they are farther from the light than their corresponding shadow-map depths.

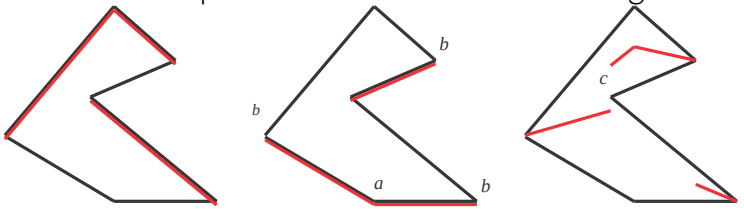


A constant depth bias is subtracted from each sample, placing each closer to the light. The blue sample is still considered in shadow because it is not closer to the light than the shadow-map depth it is tested against.



The shadow map is formed by moving each polygon away from the light proportional to its slope. All sample depths are now closer than their shadow-map depths, so all are lit.

Shadow-map surfaces for an overhead light source



front faces

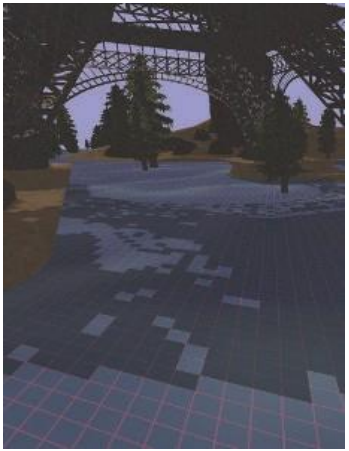
second-depth

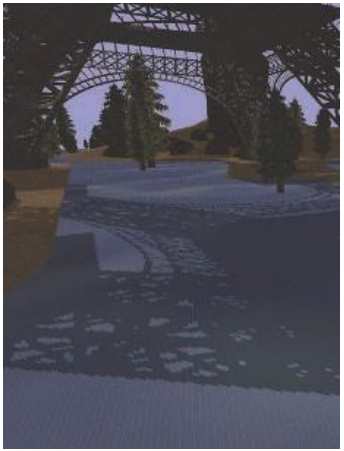
midpoint

On the left, surfaces facing the light, marked in red, are sent to the shadow map. Surfaces may be incorrectly determined to shadow themselves (“acne”), so need to be biased away from the light.

In the middle, only the backfacing triangles are rendered into the shadow map. A bias pushing these occluders downward could let light leak onto the ground plane near location a; a bias forward can cause illuminated locations near the silhouette boundaries marked b to be considered in shadow.

On the right, an intermediate surface is formed at the midpoints between the closest front- and backfacing triangles found at each location on the shadow map. A light leak can occur near point c (which can also happen with second-depth shadow mapping), as the nearest shadow-map sample may be on the intermediate surface to the left of this location, and so the point would be closer to the light.

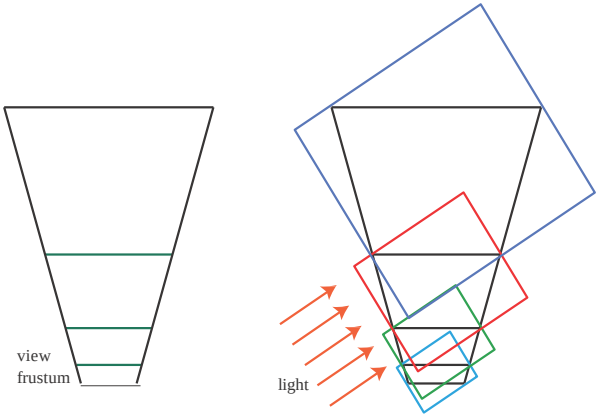




- The image to the left is created using standard shadow mapping; the image to the right using LiSPSM(light space perspective shadow maps).
- The projections of each shadow map's texels are shown.
- The two shadow maps have the same resolution, the difference being that LiSPSM reforms the light's matrices to provide a higher sampling rate nearer the viewer.

On the left the light is nearly overhead. The edge of the shadow is a bit ragged due to a low resolution compared to the eye's view.

On the right the light is near the horizon, so each shadow texel covers considerably more screen area horizontally and so gives a more jagged edge.



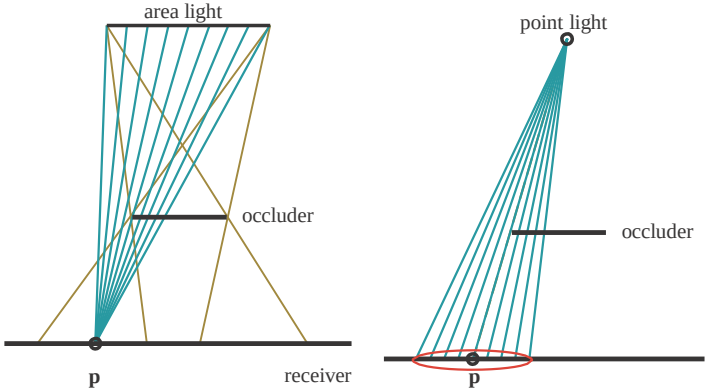
On the left, the view frustum from the eye is split into four volumes. On the right, bounding boxes are created for the volumes, which determine the volume rendered by each of the four shadow maps for the directional light.

On the left, the scene's wide viewable area causes a single shadow map at a 2048×2048 resolution to exhibit perspective aliasing.
On the right, four 1024×1024 shadow maps placed along the view axis improve quality considerably. A zoom of the front corner of the fence is shown in the inset red boxes.

Effect of depth bounds. On the left, no special processing is used to adjust the near and far planes. On the right, SDSM is used to find tighter bounds. Note the window frame near the left edge of each image, the area beneath the flower box on the second floor, and the window on the first floor, where undersampling due to loose view bounds causes artifacts. Exponential shadow maps are used to render these particular images, but the idea of improving depth precision is useful for all shadow map techniques.

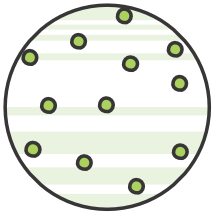
percentage-closer filtering (PCF)

- Retrieving multiple samples from a shadow map and blending the results.
- Resulting an artificially soft shadow.
- The name “percentage-closer filtering” refers to the ultimate goal, to find the percentage of the samples taken that are visible to the light. This percentage is how much light then is used to shade the surface.
- Retrieving four nearest samples off the shadow map.
- Interpolate between the results of their comparisons with the surface’s depth.
- (The surface’s depth is compared separately to the four texel depths, and for the results, 0 for shadow and 1 for light, and are then bilinearly interpolated to calculate how much the light actually contributes to the surface location.)



On the left, the brown lines from the area light source show where penumbrae are formed. For a single point p on the receiver, the amount of illumination received could be computed by testing a set of points on the area light's surface and finding which are not blocked by any occluders

On the right, a point light does not cast a penumbra. PCF approximates the effect of an area light by reversing the process: At a given location, it samples over a comparable area on the shadow map to derive a percentage of how many samples are illuminated. The red ellipse shows the area sampled on the shadow map. Ideally, the width of this disk is proportional to the distance between the receiver and occluder.



The upper left shows PCF sampling in a 4×4 grid pattern, using nearest neighbor sampling.

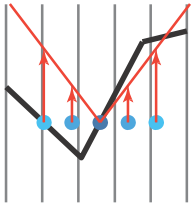
The upper right shows a 12-tap Poisson sampling pattern on a disk.

Using this pattern to sample the shadow map gives the improved result in the lower left, though artifacts are still visible.

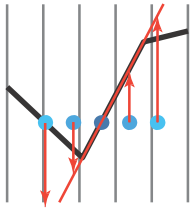
In the lower right, the sampling pattern is rotated randomly around its center from pixel to pixel. The structured shadow artifacts turn into (much less objectionable) noise.

Additional shadow bias methods

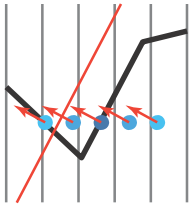
- For PCF, several samples are taken surrounding the original sample location, the center of the five dots. All these samples should be lit.



- In the left figure, a bias cone is formed and the samples are moved up to it. The cone's steepness could be increased to pull the samples on the right close enough to be lit, at the risk of increasing light leaks from other samples elsewhere (not shown) that truly are shadowed.



- In the middle figure, all samples are adjusted to lie on the receiver's plane. This works well for convex surfaces but can be counterproductive at concavities, as seen on the left side.

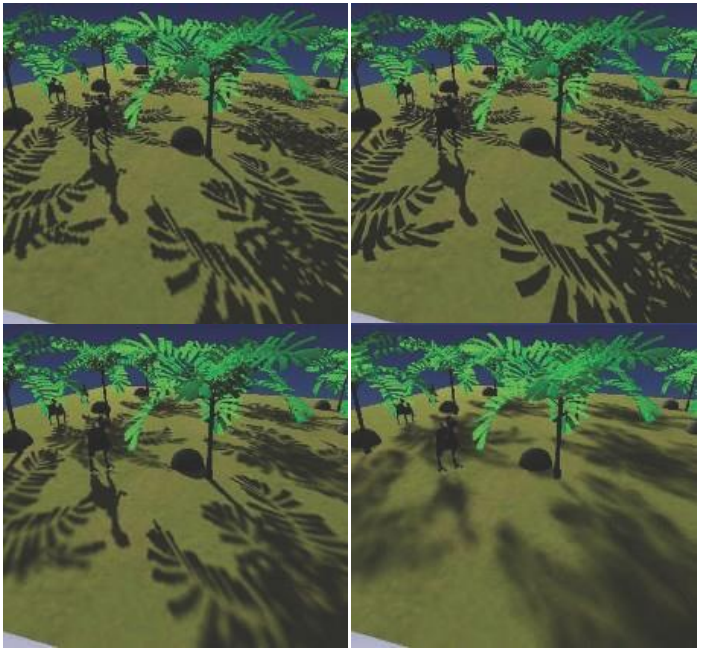


- In the right figure, normal offset bias moves the samples along the surface's normal direction, proportional to the sine of the angle between the normal and the light. For the center sample, this can be thought of as moving to an imaginary surface above the original surface. This bias not only affects the depth but also changes the texture coordinates used to test the shadow map.

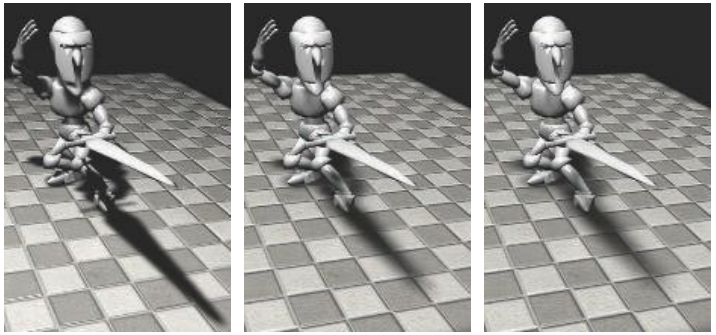
Percentage-closer filtering and percentage-closer soft shadows

- One problem with PCF is that because the sampling area's width remains constant, shadows will appear uniformly soft, all with the same penumbra width.
- This may be acceptable under some circumstances, but appears incorrect where there is ground contact between the occluder and receiver.

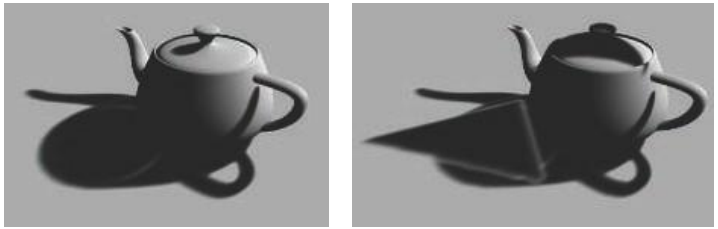
- On the left, hard shadows with a little PCF filtering.
- In the middle, constant width soft shadows.
- On the right, variable-width soft shadows with proper hardness where objects are in contact with the ground.



- In the upper left, standard shadow mapping.
- Upper right, perspective shadow mapping, increasing the density of shadow-map texel density near the viewer.
- Lower left, percentage-closer soft shadows, softening the shadows as the occluder's distance from the receiver increases.
- Lower right, variance shadow mapping with a constant soft shadow width, each pixel shaded with a single variance map sample.



- Variance shadow mapping, where the distance to the light source increases from left to right.

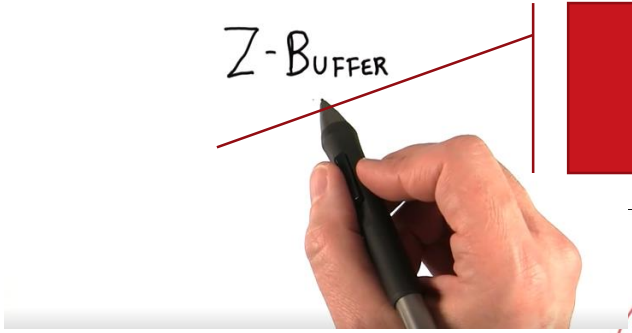


- On the left, variance shadow mapping applied to a teapot.
- On the right, a triangle (not shown) casts a shadow on the teapot, causing objectionable artifacts in the shadow on the ground.

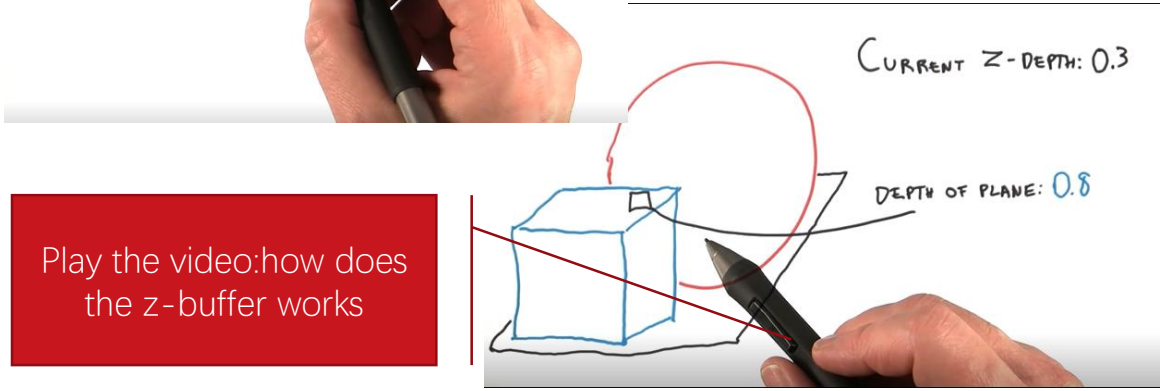
z-buffering

- known as depth buffering, it is the management of image depth coordinates in 3D graphics, usually done in hardware, sometimes in software.
- It is one solution to the visibility problem, which is the problem of deciding which elements of a rendered scene are visible, and which are hidden.
- A z-buffer can refer to a data structure or to the method used to perform operations on that structure.
- In a 3d-rendering engine, when an object is projected on the screen, the depth (z-value) of a generated pixel in the projected screen image is stored in a buffer (the z-buffer or depth buffer). A z-value is the measure of the perpendicular distance from a pixel on the projection plane to its corresponding 3d-coordinate on a polygon in world-space.
- The z-buffer has the same internal data structure as an image, namely a 2d-array, with the only difference being that it stores a z-value for each screen pixel instead of pixel data.

z-buffering and how does it works



Play the video:
Z-buffer

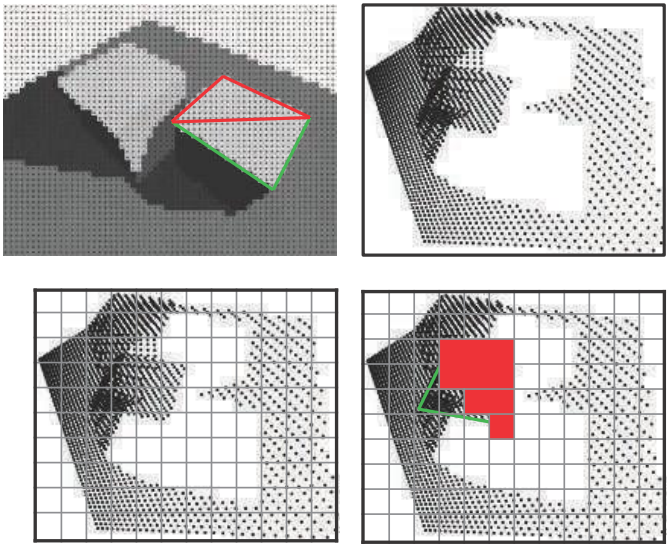


Play the video:how does the z-buffer works

irregular Z-buffer (IZB)

- an algorithm designed to solve the visibility problem in real-time 3d computer graphics.
- related to the classical Z-buffer in that it maintains a depth value for each image sample and uses these to determine which geometric elements of a scene are visible.
- the buffer itself has a normal ,regular shape for a shadow map.
- the contents are irregular, as each shadow map texel will have one or more receiver locations stored in it, or possibly none at all.
- key difference:
  - classical Z-buffer requires samples to be arranged in a regular grid.
  - irregular Z-buffer allows arbitrary placement of image samples in the image place.
- These depth samples are explicitly stored in a two-dimensional spatial data structure. During rasterization, triangles are projected onto the image plane as usual, and the data structure is queried to determine which samples overlap each projected triangle. Finally, for each overlapping sample, the standard Z-compare and (conditional) frame buffer update are performed.

Irregular z-buffer



- In the upper left, the view from the eye generates a set of dots at the pixel centers. Two triangles forming a cube face are shown.
- In the upper right, these dots are shown from the light's view.
- In the lower left, a shadow-map grid is imposed. For each texel a list of all dots inside its grid cell is generated.
- In the lower right, shadow testing is performed for the red triangle by conservatively rasterizing it. At each texel touched, shown in light red, all dots in its list are tested against the triangle for visibility by the light.

Irregular z-buffer Shadows

- On the left, PCF gives uniformly softened shadows for all objects. In the middle, PCSS softens the shadow with distance to the occluder, but the tree branch shadow overlapping the left corner of the crate creates artifacts. On the right, sharp shadows from IZB blended with soft from PCSS give an improved result

- At the top is an image generated with a basic soft-shadows approximation. At the bottom is voxel-based area light shadowing using cone tracing, on a voxelization of the scene. Note the considerably more diffuse shadows for the cars. Lighting also differs due to a change in the time of day.

Official GeForce RTX Real-Time Ray Tracing Demo

Play the video:
Atomic Heart: Official GeForce RTX Real-Time Ray Tracing Demo

Official GeForce RTX Real-Time Ray Tracing Demo

Play the video: Battlefield V: Official GeForce RTX Real-Time Ray Tracing Demo

Further Reading and Resources

- A website: https://www.khanacademy.org/partner-content/pixar/rendering/
- https://developer.nvidia.com/rtx/raytracing
- The book **Real-Time Shadows** by Eisemann et al. focuses directly on interactive rendering techniques, discussing a wide range of algorithms along with their strengths and costs.
- Woo and Poulin's **book Shadow Algorithms Data Miner** provides an overview of a wide range of shadow algorithms for interactive and batch rendering.