Computer Graphics

- Introduction to Ray Tracing -

Alexander Rath
Philipp Slusallek
Slides by Piotr Danilewski
Rendering

3D scene → camera → 2D image
Ingredients: 3D scene
Set of objects in $\mathbb{R}^3$ defined by:
Rendering

**Ingredients:** 3D scene

Set of objects in $\mathbb{R}^3$ defined by:

- Shape:
  - primitives: spheres, boxes, triangles, ...
  - implicit functions: quadrics, noise functions, ...
  - boolean operations on other shapes
  - ...

![Images of 3D shapes representing primitives and implicit functions.](image-url)
Rendering

**Ingredients:** 3D scene

Set of objects in $\mathbb{R}^3$ defined by:

- Shape:
  - primitives: spheres, boxes, triangles, ...
  - implicit functions: quadrics, noise functions, ...
  - boolean operations on other shapes
  - ...

- Material: light reflectance and emission
  - functions: diffuse, specular
  - texture
  - noise functions
  - transparency properties
Rendering

**Ingredients:** 3D scene

Set of objects in $\mathbb{R}^3$ defined by:

- **Shape:**
  - primitives: spheres, boxes, triangles, ...
  - implicit functions: quadrics, noise functions, ...
  - boolean operations on other shapes
  - ...

- **Material:** light reflectance and emission
  - functions: diffuse, specular
  - texture
  - noise functions
  - transparency properties

- **Advanced objects:**
  - volumes
  - point clouds
  - ...

Ingredients: camera
Defined in $\mathbb{R}^3$ by:
- Type:
  » perspective, orthographic, fisheye ...
- Parameters:
  » origin, direction, field-of-view ...

Perspective Orthographic
Rendering

Typical assumptions:

- Light reflected only off surfaces, objects
- Empty space is transparent
- No quantum effects
- No relativistic effects
Rendering algorithms

- Ray Tracing
  - Physically-based simulation of light transport
  - Deep recursion
  - Many effects supported out of the box
  - Slow, if no care taken
Rendering algorithms

- Rasterization
  » Imperative drawing of scene
    • Projecting whole objects
    • Shading the produced shapes
  » Shallow recursion
  » Poor support for effects
  » Fast
RAY-TRACING PRINCIPLES
Ray Tracing Is…

• Fundamental rendering algorithm
  – Simulates physical behavior of light

• Automatic, simple and intuitive
  – Easy to understand and implement
  – Delivers “correct“ images by default

• Powerful and efficient
  – Many optical global effects
  – Shadows, reflections, refractions, …
  – Efficient real-time implementation in SW and HW
  – Can work in parallel and distributed environments
  – Logarithmic scalability with scene size: O(log n) vs. O(n)
  – Output sensitive and demand driven

• Concept of light rays is not new
  – Empedocles (492-432 BC), Renaissance (Dürer, 1525), …
  – Uses in lens design, geometric optics, …
Light Transport

- Light Distribution in a Scene
  - Dynamic equilibrium
Light Transport

- **Light Distribution in a Scene**
- **Forward Light Transport**
  - from the light sources
  - reflect at surfaces
  - record when camera is hit
  - **Particle Tracing**

Most photons will not reach the camera
Intermediate results useful in more advanced algorithms
Light Transport

- Light Distribution in a Scene
- Forward Light Transport
- Backward Light Transport
  - backward from the camera
  - reflect at surfaces
  - record when light source is hit
  - Ray Tracing
    Shoot shadow rays to hit light explicitly
    Shoot more rays to find more paths and light sources
Ray Tracing Pipeline

Ray Generation
- Ray Traversal
- Intersection
- Shading
- Pixel Color

primary ray

generator
Ray Tracing Pipeline

- Ray Generation
- Ray Traversal
- Intersection
- Shading
- Pixel Color

Generator

Primary ray
Ray Tracing Pipeline

- Ray Generation
- Ray Traversal
- Intersection
- Shading
- Pixel Color

Traverse index structure find candidate

Primary ray
Ray Tracing Pipeline

Ray Generation → Ray Traversal → Intersection → Shading → Pixel Color

intersect blue box?
Ray Tracing Pipeline

- Ray Generation
- Ray Traversal
- Intersection
- Shading
- Pixel Color

- primary ray
- traverse index structure
- find next candidate
Ray Tracing Pipeline

Ray Generation

Ray Traversal

Intersection

Shading

Pixel Color

primary ray

intersect green box?
Ray Tracing Pipeline

Ray Generation
Ray Traversal
Intersection
Shading
Pixel Color

box material?
Ray Tracing Pipeline

- Ray Generation
- Ray Traversal
- Intersection
- Shading
- Pixel Color

- shadow ray
- secondary ray
Ray Tracing Pipeline

Ray Generation

Ray Traversal

Intersection

Shading

Pixel Color

light source occluded

secondary ray
Ray Tracing Pipeline

Ray Generation → Ray Traversal → Intersection → Shading → Pixel Color

box material?
Ray Tracing Pipeline
Ray Tracing Pipeline
Ray Tracing Pipeline

Ray Generation
Ray Traversal
Intersection
Shading
Pixel Color

emission?
Ray Tracing Pipeline

Ray Generation → Ray Traversal → Intersection → Shading → Pixel Color

1. Ray Generation
2. Ray Traversal
3. Intersection
4. Shading
5. Pixel Color
Ray Tracing Algorithm

```
render(camera, scene)
    foreach pixel in image
        ray = camera.generatePrimaryRay(pixel)
        color = trace(ray, scene)
        image[pixel] = color
    return image
```
Ray Tracing Algorithm

```python
trace(scene, ray):
    hit = findIntersection(scene, ray)
    return shade(scene, ray, hit.coord, hit.obj)

findIntersection(scene, ray):
    bestHit = {none, infinite}
    foreach obj in scene:
        hit = obj.intersect(ray)
        if hit succesful
            if hit.dist < bestHit.dist
                bestHit = hit
    return bestHit
```
Ray Tracing Algorithm

```python
def shade(scene, ray, coord, obj):
    material = obj.material
    color = material.emission

    for light in scene.lights:
        shadowray = light - hit
        if shadowtrace(scene, shadowray, light):
            color += light.radianceAt(hit) * material.reflectance

    for secondaryRay in material.generateSecondaryRays():
        irradiance = trace(scene, secondaryRay)
        color += irradiance * material.reflectance

    return color

def shadowtrace(scene, ray, light):
    hit = scene.findIntersection(ray)
    return (hit before light)
```
Ray Tracing Algorithm

camera.generatePrimaryRay

obj.intersect(ray)

material.emission

light.radianceAt

material.reflectance

material.generateSecondaryRays
Ray Tracing Features

• **Incorporates into a single framework**
  - Hidden surface removal
    • Front to back traversal
    • Early termination once first hit point is found
  - Shadow computation
    • Shadow rays/ shadow feelers are traced between a point on a surface and a light source
  - Exact simulation of some light paths
    • Reflection (reflected rays at a mirror surface)
    • Refraction (refracted rays at a transparent surface, Snell’s law)

• **Limitations**
  - Many reflections (exponential increase in number of rays)
  - Indirect illumination requires many rays to sample all incoming directions
  - Easily gets inefficient for full global illumination computations
  - Solution: Pick a single secondary ray at random (Monte Carlo)
    • Problem: Introduces noise that can require many samples to vanish
Ray Tracing Can...

• Produce Realistic Images
  – By simulating light transport
What is Possible?

• Models Physics of Global Light Transport
  – Dependable, physically-correct visualization
Realistic Visualization: VR/AR
Lighting Simulation
What is Possible?

- **Huge Models**
  - Logarithmic scaling in scene size

12.5 Million Triangles

~1 Billion Triangles
Outdoor Environments

$90 \times 10^{12}$ (trillion) triangles
Boeing 777: ~350 million individual polygons, ~30 GB on disk
Volume Visualization

Iso-surface rendering
Games?
Games!
Ray Tracing in CG

- **In the Past (until end of 80ies)**
  - Was computationally very demanding (minutes to hours per frame)
  - Tried hard to speed it up, but always too slow → only off-line use
- **“Lost generation” (1990ies)**
  - Believed ray tracing would not be suitable for HW implementations
  - Believed ray tracing would always be slower than rasterization
- **More Recently**
  - Interactive ray tracing on supercomputers [Parker, U. Utah‘98]
  - Interactive ray tracing on PCs [Wald‘01]
  - Distributed real-time ray tracing on PC clusters [Wald‘01]
  - RPU: First full HW implementation [Siggraph 2005]
  - Commercial tools: Embree (Intel/CPU), OptiX (Nvidia/GPU)
  - Complete film industry has switched to ray tracing (Monte-Carlo)
- **Own conference**
  - Symposium on Interactive RT, now High-Performance Graphics (HPG)
- **Ray tracing systems**
  - Research: PBRT (offline, physically-based, based on book, OSS), Mitsuba-2 renderer (EPFL), Rodent (SB), ...
  - Products: Blender (OSS), V-Ray (Chaos Group), Arnold & VRED (Autodesk), Corona (Render Legion), MentalRay/iRay (MI), ...
Ray Casting Outside CG

• **Tracing/Casting a ray**
  – Special type of query
    • “Is there a primitive along a ray”
    • “How far is the closest primitive”

• **Other uses than rendering**
  – Visibility computation
  – Volume computation
  – Collision detection
  – Acoustics
  – Radar
  – …