Welcome to 02941: Physically Based Rendering and Material Appearance Modelling

Jeppe Revall Frisvad

June 2023

Course responsible

- Jeppe Revall Frisvad
  - Associate Professor, DTU Compute
  - https://people.compute.dtu.dk/jerf/
  - jerf@dtu.dk
  - Lectures and exercises

Course contents

Core elements:

- Radiative transfer.
  - Visual effects: emission, diffuse and rough surface reflection, shadows, indirect illumination (colour bleeding), caustics, participating media, translucency.
  - Methods: path tracing, photon mapping, diffusion.
- Geometrical optics.
  - Visual effects: reflection, refraction, absorption, dispersion, polarisation.
  - Methods: path tracing, photon mapping, wave theory (refractive index, Fresnel).
- Light scattering.
  - Visual effects: interference, diffraction, scattering by particles and microgeometry.
  - Methods: Computing reflectance distribution functions and scattering properties.

Assessment

- Daily exercises.
  - Each worksheet has *deliverables* which are part of your assessment.
    Think of it as your lab journal.
  - Your work should be collected in a pdf and submitted before the final deadline:
    23:59 Friday 23 June 2023.
- One slide displaying results from the lab journal and/or project.
  Presentation the last day, submission the day after.
- Your work is assessed in its entirety and you will receive a *pass* or *not pass* grade.

02941 Physically Based Rendering

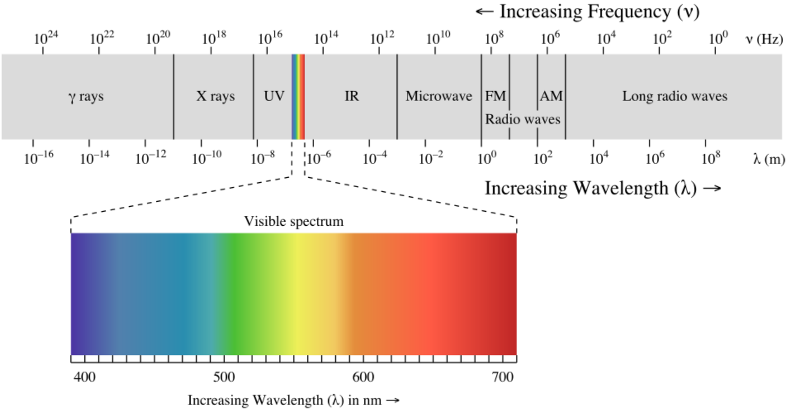
Introduction

Jeppe Revall Frisvad

June 2023

Quiz: What is the origin of colours?

- Waves of light have different wavelengths which are perceived as different colours.



- Light from the sun is white (contains all wavelengths), how come other colours appear in nature. . .

Quiz: Why are leaves green?

Quiz: Why are metals shiny, but not perfect mirrors?

https://en.wikipedia.org/wiki/Copper

Quiz: Why is lava red-hot?

https://en.wikipedia.org/wiki/Black-body_radiation

Quiz: Why is the sky blue, but red at sunset?

Quiz: Why rainbows?

https://people.compute.dtu.dk/jerf/papers/on_LL.pdf

Quiz: Why are soap bubbles multicoloured?

https://www.soapbubble.dk/

What is physically based rendering?

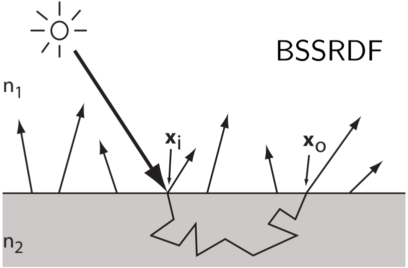
- **Rendering**: the particular way in which something is performed. (Oxford Advanced Learner's Dictionary)
- **Rendering an image**: the particular way in which an image is generated.
- **Photographic rendering**: the particular way in which an image is generated using a camera (including development).
- **Computer graphics rendering**: the particular way in which an image is generated using a computer.
- **Physically based rendering**: a physically based way of computing an image.
  - Think of a photographic rendering as a physical experiment.
  - Physically based rendering is then an attempt to model photographic rendering mathematically and computationally.
  - The (unreachable) goal of the models is to predict the outcome of the physical experiment: "taking a picture".

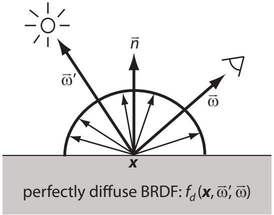
Models needed for physically based rendering

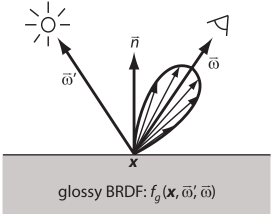
- Consider the experiment: “taking a picture”.
- What do we need to model it?
  - Camera
  - Scene geometry
  - Light sources
  - Light propagation
  - Light absorption and scattering
- Mathematical models for these physical phenomena are required as a minimum in order to render an image.
- We can use very simple models, but, if we desire a high level of realism, more complicated models are required.
- To get started, we will recall the simpler models (in opposite order).

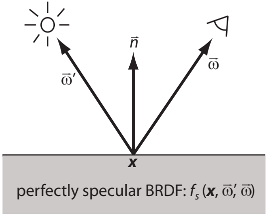
Materials (light scattering and absorption)

- Optical properties (index of refraction, $n(\lambda) = n'(\lambda) + i\, n''(\lambda)$).
- Reflectance distribution functions, $S(\boldsymbol{x}_i, \vec{\omega}_i; \boldsymbol{x}_o, \vec{\omega}_o)$.



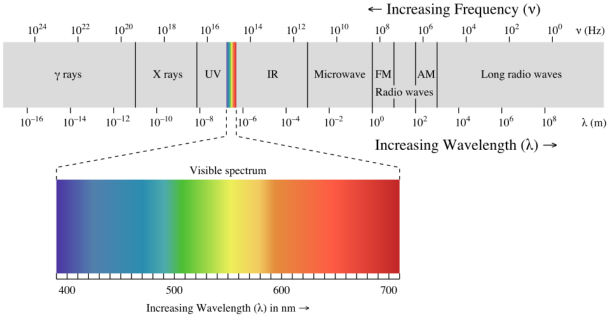






Light propagation

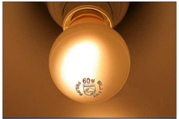
- Visible light is electromagnetic waves of wavelengths (λ) from 380 nm to 780 nm.

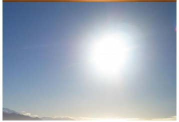


- Electromagnetic waves propagate as *rays of light* for $\lambda \to 0$.
- Rays of light follow the path of least time (Fermat).
- How does light propagate in air? In straight lines (almost).
- The parametrisation of a straight line in 3D ($\boldsymbol{r}(t) = \boldsymbol{x} + t\vec{\omega}$) is therefore a good, simple model for light propagation.

Light sources

- A light source is described by a spectrum of light $L_{e,\lambda}(\boldsymbol{x}, \vec{\omega}_o)$ which is emitted from each point on the emissive object.
- A simple model is a light source that from each point emits the same amount of light in all directions and at all wavelengths, $L_{e,\lambda} = \mathrm{const}$.
- The spectrum of heat-based light sources can be estimated using Planck's law of radiation. Examples:



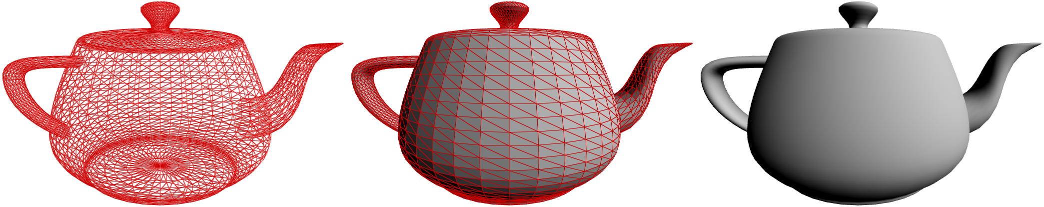



- The surface geometry of light sources is modelled in the same way as other geometry in the scene.

Scene geometry

- Surface geometry is often modelled by a collection triangles some of which share edges (a triangle mesh).
- Triangles provide a discrete representation of an arbitrary surface.

  Teapot example:



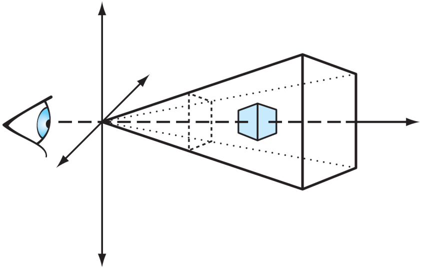
wireframe faces shaded

- Triangles are useful as they are defined by only three vertices.

  And ray-triangle intersection is simple.

Camera

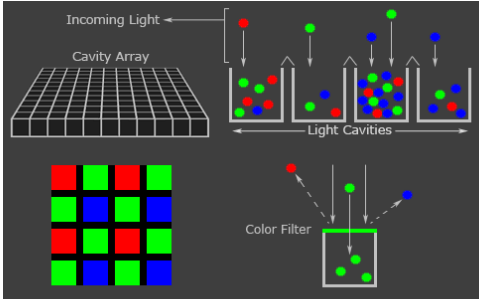
- A camera consists of a light sensitive area, a processing unit, and a storage for saving the captured images.
- The simplest model of a camera is a rectangle, which models the light sensitive area (the chip/film), placed in front of an eye point where light is gathered.



- We can use this model in two different ways:
  - Follow rays from the eye point through the rectangle and onwards (ray casting).
  - Project the geometry on the image plane and find the geometry that ends up in the rectangle (rasterization).

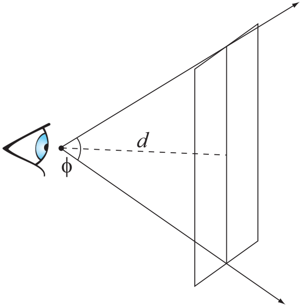
The light sensitive Charge-Coupled Device (CCD) chip

- A CCD chip is an array of light sensitive cavities.
- A digital camera therefore has a resolution $W \times H$ measured in number of pixels.
- A pixel corresponds to a small area on the chip.
- Several light sensitive cavities contribute to each pixel because the light measurement is divided into red, green, and blue.
- Conversion from this colour pattern to an RGB image is called demosaicing.



The lens as an angle and a distance

- The lens system determines how large the field of view is.
- The field of view is an angle ϕ.



$$h = 2d \tan\left(\frac{\phi}{2}\right)$$

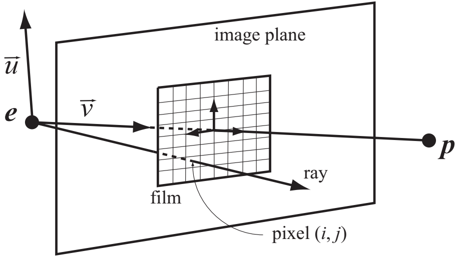
- The lens also determines the distance d from the eye point to the image plane wherein the light sensitive area is placed in the model.
- The distance d is called the camera constant.
- Since the size of the chip is constant, d determines the zoom level of the camera.

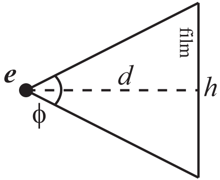
Ray generation

- Camera description:

Extrinsic parameters		Intrinsic parameters	
$\boldsymbol{e}$	Eye point	ϕ	Vertical field of view
$\boldsymbol{p}$	View point	d	Camera constant
$\vec{u}$	Up direction	W, H	Camera resolution

- Sketch of ray generation:





- Given pixel index (i, j), we find the direction $\vec{\omega}$ of a ray through that pixel.

02941 Physically Based Rendering

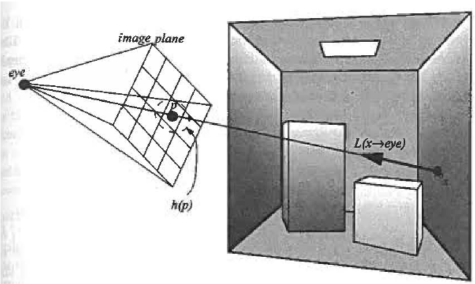
Ray tracing direct illumination

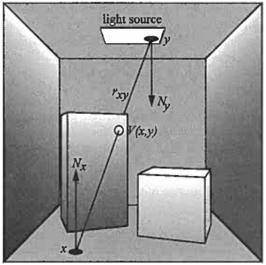
Jeppe Revall Frisvad

June 2023

What is a ray?

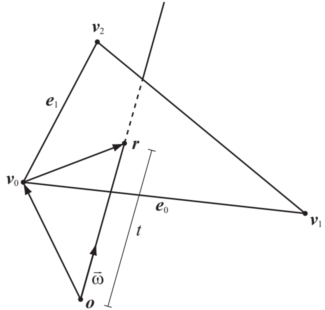
- Parametrisation of a straight line: $\boldsymbol{r}(t) = \boldsymbol{e} + t\,\vec{\omega} \quad , \quad t \in [0, \infty)$.
- Camera provides origin ($\boldsymbol{e}$) and direction ($\vec{\omega}$) of “eye rays”.





- The user sets origin and direction when tracing rays recursively.
- But we need more properties:
  - Minimum and maximum distances (t_{min} and t_{max}) for numerics and visibility.
  - Info on what was hit and where (hit normal, position, distance, material, etc.).
  - A counter to tell us the trace depth: how many reflections and refractions in a path (no. of recursions).

Ray-triangle intersection



- Ray: $\boldsymbol{r}(t) = \boldsymbol{o} + t\,\vec{\omega},\ \ t \in [t_{\min}, t_{\max}]$.
- Triangle: $\boldsymbol{v}_0,\ \ \boldsymbol{v}_1,\ \ \boldsymbol{v}_2$.
- Edges and normal:

  $\boldsymbol{e}_0 = \boldsymbol{v}_1 - \boldsymbol{v}_0,\ \ \boldsymbol{e}_1 = \boldsymbol{v}_0 - \boldsymbol{v}_2,\ \ \boldsymbol{n} = \boldsymbol{e}_0 \times \boldsymbol{e}_1$.
- Barycentric coordinates:

$$\begin{aligned} \boldsymbol{r}(u,v,w) &= u\boldsymbol{v}_0 + v\boldsymbol{v}_1 + w\boldsymbol{v}_2 = (1-v-w)\boldsymbol{v}_0 + v\boldsymbol{v}_1 + w\boldsymbol{v}_2 \\ &= \boldsymbol{v}_0 + v\boldsymbol{e}_0 - w\boldsymbol{e}_1\ . \end{aligned}$$

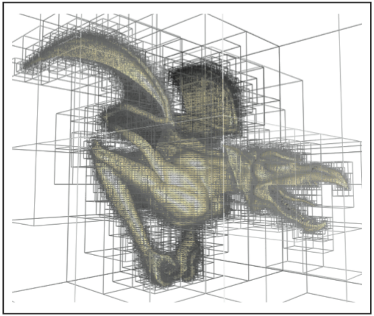
- The ray intersects the triangle's plane at $t' = \dfrac{(\boldsymbol{v}_0 - \boldsymbol{o}) \cdot \boldsymbol{n}}{\vec{\omega} \cdot \boldsymbol{n}}$.
- Find $\boldsymbol{r}(t') - \boldsymbol{v}_0$ and decompose it into portions along the edges $\boldsymbol{e}_0$ and $\boldsymbol{e}_1$ to get v and w. Then check

$$v \geq 0 \quad , \quad w \geq 0 \quad , \quad v + w \leq 1\ .$$

Spatial subdivision

- To model arbitrary geometry with triangles, we need many triangles.
- A million triangles and a million pixels are common numbers.
- Testing all triangles for all pixels requires 10^{12} ray-triangle intersection tests.
- If we do a million tests per millisecond, it will still take more than 15 minutes.
- This is prohibitive. We need to find the relevant triangles.

- Spatial data structures offer logarithmic complexity instead of linear.
- A million tests become twenty operations $\left(\log_2 10^6 \approx 20\right)$.
- 15 minutes become 20 milliseconds.



Gargoyle embedded in oct tree [Hughes et al. 2014].

Ray tracing

- What do you need in a ray tracer?
  - Camera (ray generation and lens effects)
  - Ray-object intersection (and accelleration)
  - Light distribution (different source types)
  - Visibility testing (for shadows)
  - Surface scattering (reflection models)
  - Recursive ray tracing (rays spawn new rays)
- How to use a ray tracer? Trace radiant energy.
- The energy travelling along a ray of direction $\vec{r} = -\vec{\omega}$ is measured in radiance (flux per projected area per solid angle).
- The outgoing radiance L_o at a surface point $\boldsymbol{x}$ is the sum of emitted radiance L_e and reflected radiance L_r:

$$L_o(\boldsymbol{x}, \vec{\omega}) = L_e(\boldsymbol{x}, \vec{\omega}) + L_r(\boldsymbol{x}, \vec{\omega}) \ .$$

- Reflected radiance is computed using the BRDF (f_r) and an estimate of incident radiance L_i at the surface point.

The rendering equation

- Surface scattering is defined in terms of
  - Radiance:
$$L = \frac{\mathrm{d}^2\Phi}{\cos\theta\,\mathrm{d}A\,\mathrm{d}\omega} \quad .$$
  - Irradiance:
$$E = \frac{\mathrm{d}\Phi}{\mathrm{d}A} \quad , \quad \mathrm{d}E = L_i \cos\theta\,\mathrm{d}\omega \quad .$$
  - BRDF:
$$f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_o) = \frac{\mathrm{d}L_r(\boldsymbol{x}, \vec{\omega}_o)}{\mathrm{d}E(\boldsymbol{x}, \vec{\omega}_i)} \quad .$$
- The rendering equation then emerges from $L_o = L_e + L_r$:
$$L_o(\boldsymbol{x}, \vec{\omega}_o) = L_e(\boldsymbol{x}, \vec{\omega}_o) + \int_{2\pi} f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_o)\, L_i(\boldsymbol{x}, \vec{\omega}_i)\, \cos\theta_i\,\mathrm{d}\omega_i \quad .$$
- This is an integral equation. Integral equations are recursive in nature.

Surface scattering

- Bidirectional Reflectance Distribution Functions (BRDFs)

$$f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_o) = \frac{\mathrm{d}L(\boldsymbol{x}, \vec{\omega}_o)}{\mathrm{d}E(\boldsymbol{x}, \vec{\omega}_i)} \ .$$

- Physically-based BRDFs must obey:
  - Reciprocity:

$$f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_o) = f_r(\boldsymbol{x}, \vec{\omega}_o, \vec{\omega}_i) \ .$$

  - Energy conservation:

$$\int_{2\pi} f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_o) \cos\theta_o \, \mathrm{d}\omega_o \leq 1 \ .$$

- The Lambertian (perfectly diffuse) BRDF scatters light equally in all directions

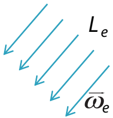
$$f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_o) = \frac{\rho_d}{\pi} \ .$$

where ρ_d is the bihemispherical diffuse reflectance ($\mathrm{d}\Phi_r / \mathrm{d}\Phi_i$).

Direct illumination due to different light sources

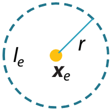
- A directional light emits a constant radiance L_e in one particular direction $\vec{\omega}_e = -\vec{\omega}_i$

$$L_r = \int_{2\pi} f_r L_i \cos\theta_i \, \mathrm{d}\omega_i = f_r \, V L_e \left(-\vec{\omega}_e \cdot \vec{n}\right).$$



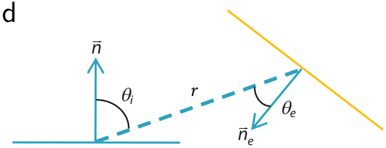
- A point light emits a constant intensity I_e in all directions from one particular point $\boldsymbol{x}_e$

$$L_r = f_r \frac{V}{r^2} \left(\vec{\omega}_i \cdot \vec{n}\right) I_e \,, \qquad \begin{aligned} r &= \|\boldsymbol{x}_e - \boldsymbol{x}\| \\ \vec{\omega}_i &= (\boldsymbol{x}_e - \boldsymbol{x})/r \end{aligned} \quad .$$



- An area light emits a cosine weighted radiance distribution from each differential element of area $\mathrm{d}A$. We have $\mathrm{d}\omega_i = \frac{\cos\theta_e}{r^2} \mathrm{d}A_e$ and

$$L_r = \int f_r \, V L_e \, \cos\theta_i \, \frac{\cos\theta_e}{r^2} \mathrm{d}A_e \,.$$



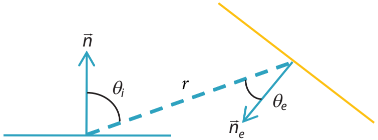
$$\cos\theta_e = -\vec{\omega}_i \cdot \vec{n}_e$$

V is visibility.

Approximation for a distant area light

- An area light emits a cosine weighted radiance distribution from each area element

$$L_r = \int f_r \, V L_e \, \cos\theta_i \, \frac{\cos\theta_e}{r^2} \mathrm{d}A_e \, .$$



$$\cos\theta_e = -\vec{\omega}_i \cdot \vec{n}_e$$

  V is visibility.

- Assuming the area light is distant, we can approximate its lighting of the scene using just one sample. Suppose we place it at the center of the bounding box $\boldsymbol{x}_e$. Then

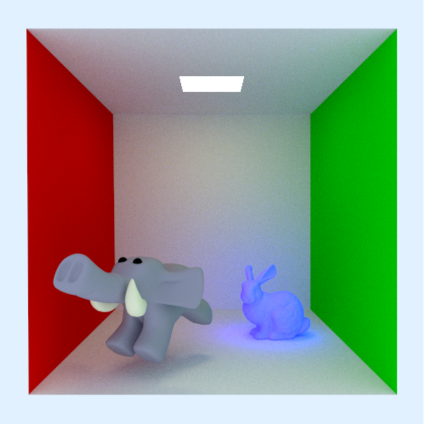
$$L_r = f_r \, \frac{V}{r^2} \, (\vec{\omega}_i \cdot \vec{n}) \underbrace{\sum_{\triangle=1}^{N} (-\vec{\omega}_i \cdot \vec{n}_{e\triangle}) L_{e\triangle} \, A_\triangle}_{I_e} \, ,$$

  where N is the number of triangles in the area light and $\triangle$ is a triangle index.

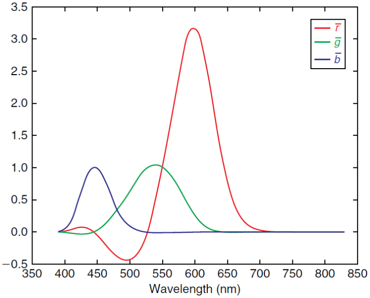
- This is like a point light, but with a different intensity.

Sampling a triangle mesh (area lights, soft shadows)

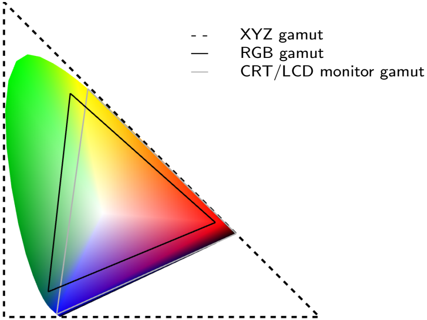
- Material:
$$f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_o) = \frac{\rho_d(\boldsymbol{x})}{\pi}\,.$$
- Sampler (triangle index Δ and area A_Δ):
$$\vec{\omega}_{i,q} = \frac{\boldsymbol{x}_{\ell,q} - \boldsymbol{x}}{\|\boldsymbol{x}_{\ell,q} - \boldsymbol{x}\|}$$
$$\mathrm{pdf}(\boldsymbol{x}_{\ell,q}) = \mathrm{pdf}(\triangle)\mathrm{pdf}(\boldsymbol{x}_{\ell,q,\triangle}) = \frac{1}{N_\Delta}\frac{1}{A_\triangle}\,.$$
- Estimator (no. of triangles N_Δ):
$$L_{r,q}(\boldsymbol{x}, \vec{\omega}_o) = f_{r,q} L_{i,q} \cos\theta_{i,q}$$
$$= \frac{\rho_d(\boldsymbol{x})}{\pi} \underbrace{L_e(\boldsymbol{x}_{\ell,q}, -\vec{\omega}_{i,q}) V(\boldsymbol{x}_{\ell,q}, \boldsymbol{x}) \frac{(-\vec{\omega}_{i,q} \cdot \vec{n}_e)}{\|\boldsymbol{x}_{\ell,q} - \boldsymbol{x}\|^2} N_\Delta A_\Delta}_{L_{i,q}} (\vec{\omega}_{i,q} \cdot \vec{n})\,.$$



Colorimetry (spectrum to RGB)



CIE color matching functions



The chromaticity diagram

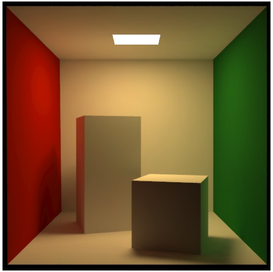
$$R = \int_{\mathscr{V}} C(\lambda)\bar{r}(\lambda)\,\mathrm{d}\lambda \quad , \quad G = \int_{\mathscr{V}} C(\lambda)\bar{g}(\lambda)\,\mathrm{d}\lambda \quad , \quad B = \int_{\mathscr{V}} C(\lambda)\bar{b}(\lambda)\,\mathrm{d}\lambda \ ,$$

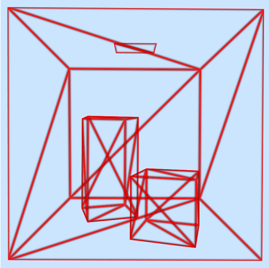
where $\mathscr{V}$ is the interval of visible wavelengths and $C(\lambda)$ is the spectrum that we want to transform to RGB.

Exercises

- Find out how to set the material properties of objects in a scene. Change the diffuse reflectance (ρ_d).
- Load triangle meshes and material properties from files.
- Ray trace loaded meshes.
- Shade Lambertian materials using a directional light.
- Shade Lambertian materials using an area light.
- Compute visibility (V) by tracing shadow rays to light sources.

The Cornell box





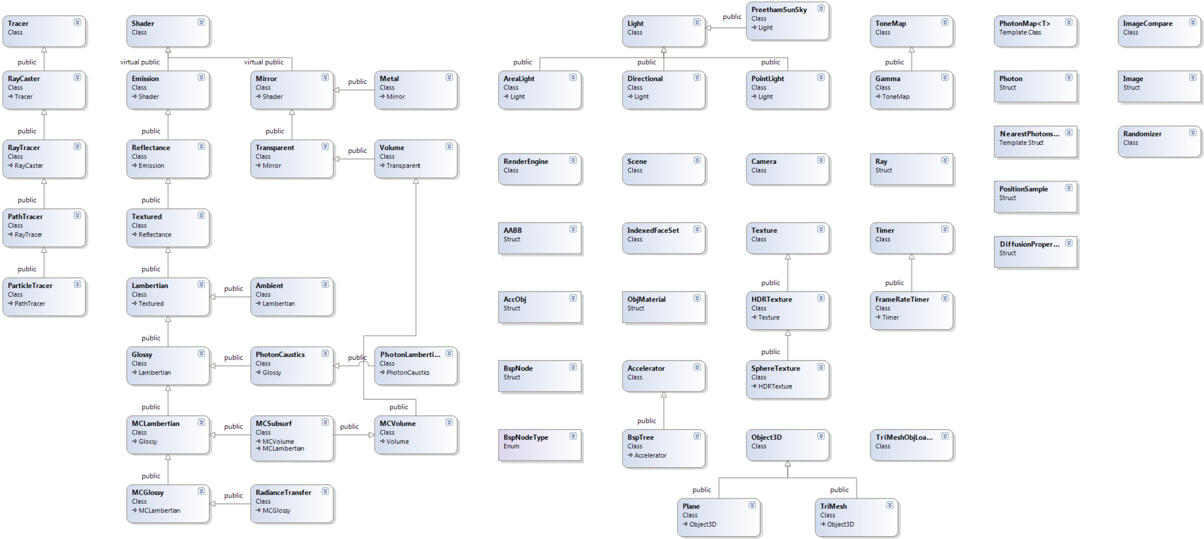
- The Cornell box is a convenient test scene for developing rendering algorithms. https://www.graphics.cornell.edu/online/box/
- You can load the Cornell box (or other .obj files) into the ray tracing framework by supplying the following commandline arguments:

CPU `../models/CornellBox.obj ../models/CornellBlocks.obj`

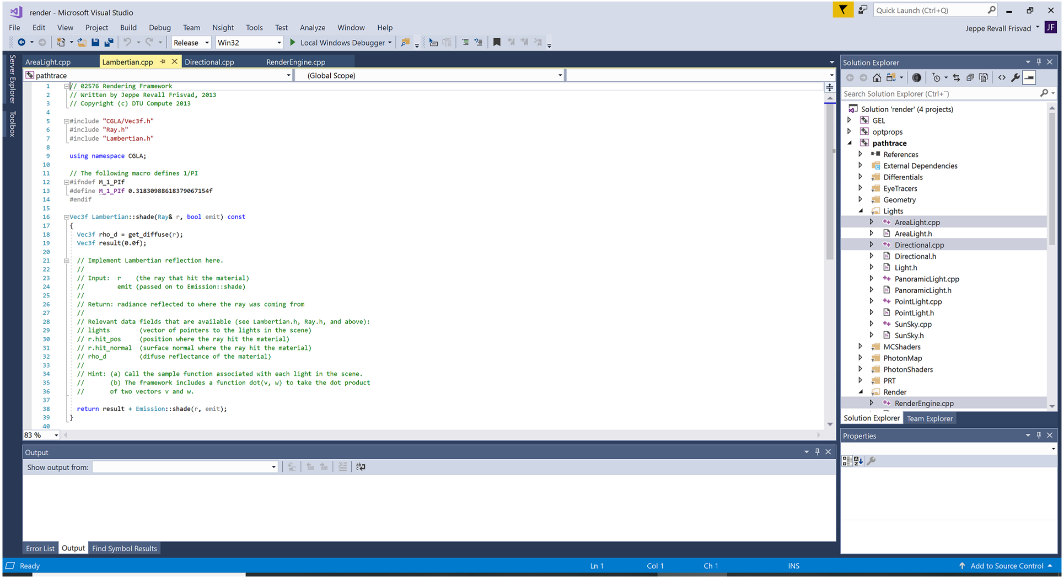
GPU `../../models/CornellBox.obj ../../models/CornellBlocks.obj`

- Loading the blocks is optional. You can load the box only and insert specular spheres to test more light paths.

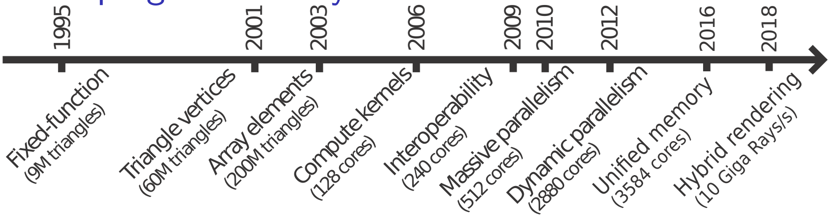
Implementing a ray tracer (Render Framework)



Files used in Worksheet 1 (Render Framework)



Timeline on the programmability of the GPU



1995 Fixed-function rasterization pipeline in hardware.

2001 Vertex shaders (first programmable part of the pipeline).

2003 Fragment/pixel shaders (GPGPU).

2006 Unified shaders (CUDA) and geometry shaders.

2008 Double precision arithmetics.

2009 Compute shaders (interoperability) and tesselation shaders.

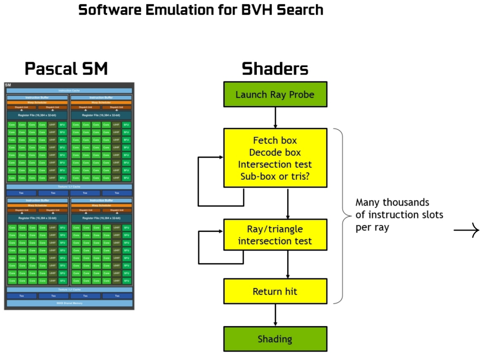
2010 Streaming multiprocessor architecture. **Programmable ray tracing pipeline on the GPU.**

2012 Dynamic parallelism (threads spawn threads).

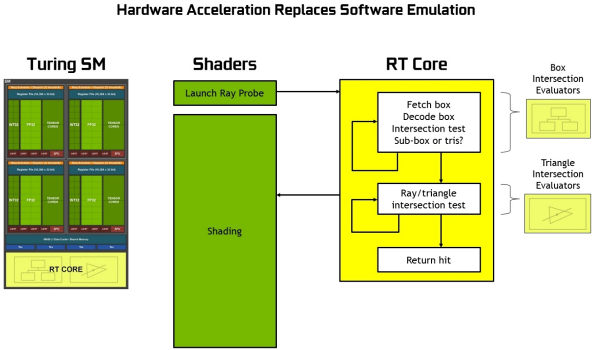
2016 Unified memory (on demand data migration and dynamic memory allocation).

2018 **Hybrid rendering** (CUDA cores, **RT cores**, DNN tensor cores).

Ray tracing with RT cores

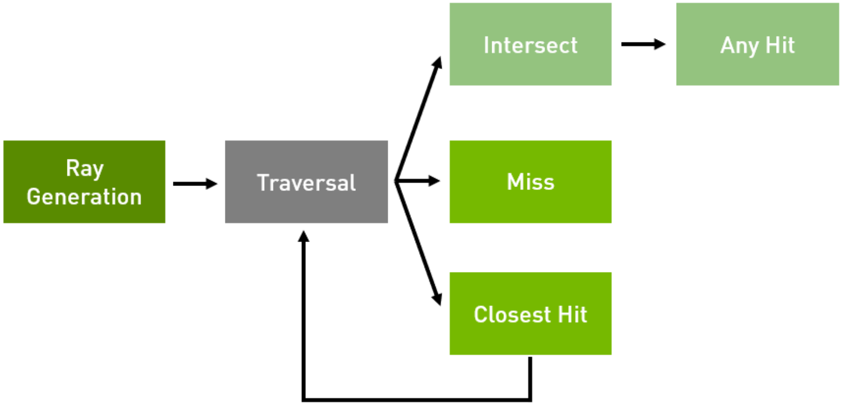


Ray tracing with CUDA cores



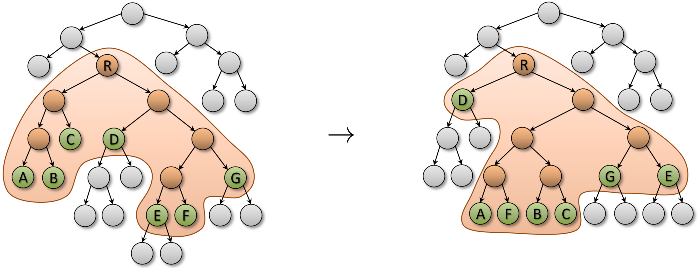
Ray tracing with CUDA and RT cores

GPU ray tracing (OptiX)



- The camera model is used for the "Ray Generation" program.
- Ray-object intersection is in the "Intersect" program (hardwired for triangles).
- The shader is implemented in the "Closest Hit" program.

Treelet restructuring bounding volume hierarchy for spatial subdivision



treelet → reorganized treelet

- Practical GPU-based bounding volume hierarchy (BVH) builder.
  1. Build a low-quality BVH (parallel linear BVH).
  2. Optimize node topology by parallel treelet restructuring (keeping leaves and their subtrees intact).
  3. Post-process for fast traversal.

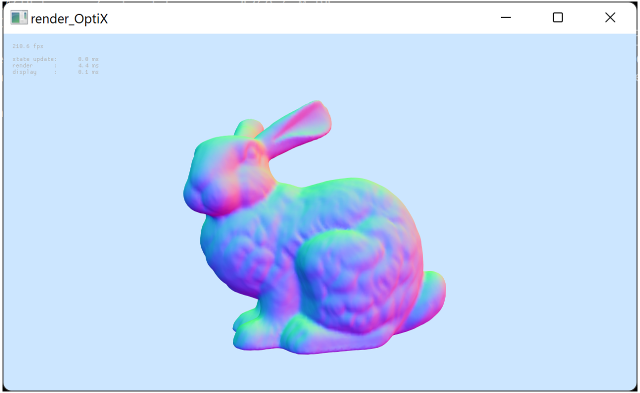
References

- Karras, T., and Aila, T. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of HPG 2013*, pp. 89–99. ACM, July 2013.

Simplest closest hit program

```
extern "C" __global__ void __closesthit__normals()
{
  const HitGroupData* hit_group_data = reinterpret_cast<HitGroupData*>(optixGetSbtDataPointer());
  const LocalGeometry geom = getLocalGeometry(hit_group_data->geometry_data);

  float3 result = normalize(geom.N)*0.5f + 0.5f;
  setPayloadResult(result);
}
```



Files used in Worksheet 1 (Render OptiX Framework)