Rendering Algorithms

Spring 2014

Matthias Zwicker

Universität Bern

Programming projects

- Description of assignments on ilias webpage
- Skeleton ray tracer in Java is provided
  - We recommend that you use it...
- But you can also create your own ray tracer from scratch!
  - You can use Java, C++, C#, etc. but no slow interpreted or scripting languages (Smalltalk, Eiffel, Python, etc.)

Programming projects

- Turn-in of assignments on ilias *and* by demonstration to teaching assistant
  - Usually during the excercise session in the CGG pool (Neubrückstrasse 10, 2nd floor)
- Grade of assignments affects final course grade!
  - (*average grade of all assignments* + *final exam grade*) / 2 = course grade!

Skeleton Ray Tracer

- Skeleton ray tracer available on github
  - https://github.com/mzwicker/Rendering-Algorithms-2014-Skeleton
- Provides some basic functionality
- Java doc available
  - Study it first!

Assignment 1: Basic ray tracer

- Topics
  - Generate camera rays
  - Intersection routines for spheres, planes, triangles
  - Computational solid geometry (CSG)
  - Meshes, instancing
  - Phenomenological shading
  - Shadows
  - Reflection, Refraction
  - Hacker's bonus: procedural shaders, textures, bump maps
- Due date: in 3 weeks (march 13th)

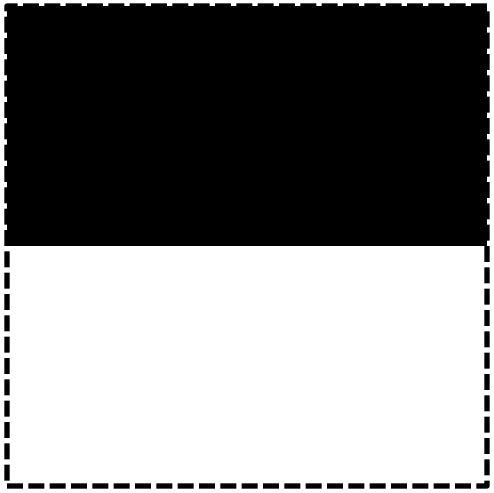
Assignment 1: Basic ray tracer

- Start now!
- 1st week
  - Understand the base code!
  - Camera rays
- 2nd week
  - Intersection with spheres, triangles, meshes
  - CSG
  - Instancing
- 3rd week
  - Shading (Blinn model + shadows)
  - Reflection and refraction
  - Hacker‘s bonus (procedural shaders, textures, bump maps, nice scenes)

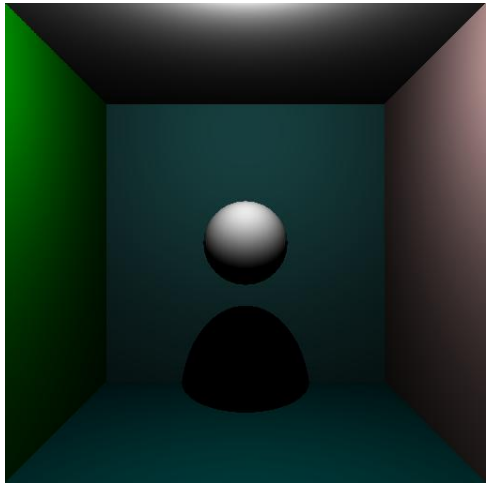
Assignment 1: Basic ray tracer

```
rayTrace() {
  construct scene representation

  for each pixel
    ray = computePrimary()
    hit = first intersection with scene
    color = shade( hit )
    set pixel color
}
```

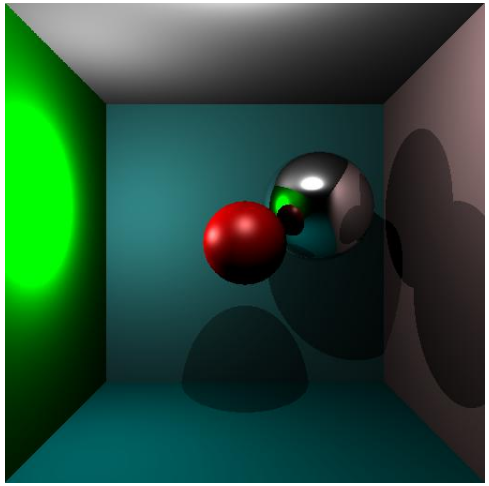
Assignment 1: Basic ray tracer



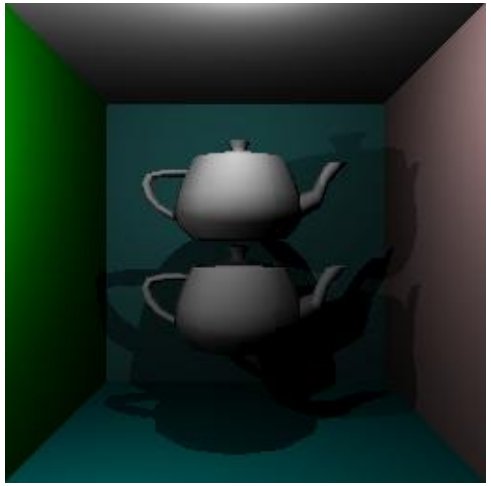
A horizontal plane!



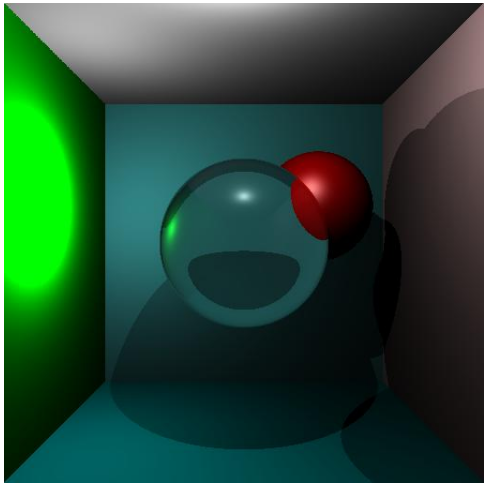
Shading, shadows



Mirrors



Meshes, instancing



Refraction

Used Java libraries

- Matrix and vector math
  - For example javax.vecmath.*
- Image input/output
  - For example java.awt.image.*, in particular class BufferedImage

Useful C++ libraries

- Boost libraries, http://www.boost.org/
  - Includes libraries for matrix math, image i/o, and many more

Skeleton ray tracer structure

```
rt

rt.basicscenes
rt.cameras
rt.film
rt.integrators
rt.intersectable
rt.lightsources
rt.materials
rt.samplers
rt.testscenes
rt.tonemappers
```

Ray tracing pseudo code

- Main loop (in Main.RenderTask.run())

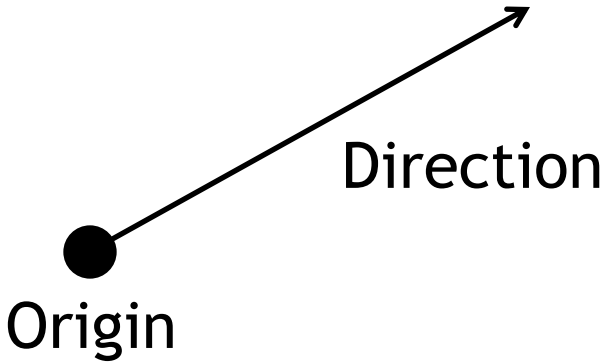
```
for all pixels in block {

    float samples[][]
          = integrator.makePixelSamples(sampler,spp);

    for all samples{
        Ray r = camera.makeWorldSpaceRay(pixel, sample);
        Spectrum s = integrator.integrate(r);
        film.addSample(pixel, s);
    }

}
```

Sampler

- Generates sequences of values that can be used by different components of the ray tracer.
- More later

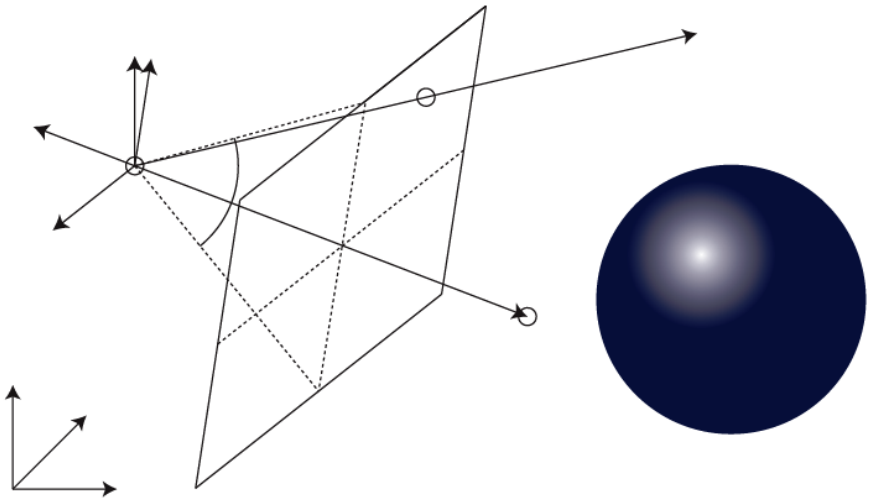
Ray

- Origin, direction



Camera & image specification

- Parameters: eye position, look-at point, up vector, vertical field of view, aspect ratio, width in pixels, height in pixels
- Make primary rays given pixel coordinates and sample (`makeWorldSpaceRay(pixel, sample)`)



Hit record

- Stores information about a ray-surface intersection
  - Position
  - Surface normal
  - Ray direction
  - Ray parameter at intersection
  - Reference to material
  - Reference to intersectable
  - (tangent vectors, texture coordintes) more later...

Material

- Stores material properties, information for shading
- More later

Integrator

- Main method `integrate`
  - Computes color for a ray
  - Implements core ray tracing algorithm
- Different integrators can use different algorithms
  - Simple integrator for debugging
  - Shading without shadows
  - Recursive ray tracing
  - Path tracing
  - Etc.

„Binary Integrator“

- For testing, debugging

```
integrate(ray)

  if(root.intersect(ray) != null)
    return white
  else
    return black
```

- DebugIntegrator works similarly (but is a bit more complex)

Integrator with light sources

- If something is hit, sum over all light sources

```
integrate(ray)

  hitRecord = root.intersect(ray)

  if( hitRecord != null )
    spectrum = black
    for all light sources
      spectrum += contribution of light source
    return spectrum
  else
    return black
```

- Look at PointLightIntegrator for details...

Intersectable

- Interface implemented by anything that can be intersected with a ray
- Implements method `Intersect(Ray r)`

Spheres and planes

- Implement interface **`Intersectable`**
- Sphere: center, radius
- Plane: distance to origin, normal
- Intersection methods as *discussed in next class*
  - Ray-plane intersection http://www.siggraph.org/education/materials/HyperGraph/raytrace/rayplane_intersection.htm
  - Ray-sphere intersection http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm
- Plane already implemented in Skeleton ray tracer, Sphere needs to be added

Aggregate

- Abstract class that stores a collection of intersectables
- Intersecting an aggregate returns the closest hit
- Abstract method `iterator()` has to iterate through the elements in the aggregate
- A scene is represented by a single aggregate (the **root**)

Mesh

- Implements `Aggregate`
- Stores a collection of triangles
  - Array of vertex positions
  - Array of vertex normals
  - Array of indices, stores indices of three triangle vertices in position, normal arrays

CSG

- `CSGNode` is alerady implemented
- More about this in the next lecture...

Mesh intersection

- Iterate over all triangles
  - Intersect each triangle
  - Retain closest hit, i.e., smallest t parameter of ray
- Return closest hit

MeshTriangle

- Stores reference to mesh
  - Contains index into index array of mesh
  - Looks up vertex data in arrays of mesh
- Implement interface **`Intersectable`**
  - Intersection method as *discussed in next class*

Spectrum, ToneMapper

- Spectrum
  - Stores r,g,b color values
- ToneMapper
  - Writes **`Film`** to an image file

Film

- Accumulates color samples into a 2D array of spectra
- Implements `addSample(pixel, spectrum)`

- Simple example that stores average of samples in pixel:

```
addSample(pixel p, spectrum s)
    if pixel inside image{
        nsamples++;
        image[p.x][p.y]
          =(image[p.x][p.y]*(nsamples-1) + s)/nsamples;
    }
```

Scene descriptions

- Scene description are kept in separate files to keep code clean
  - See example files in `rt.basicscenes`
- Some test scenes are available in `rt.testscenes`
  - They don‘t work with the provided skeleton code!
  - Use them to verify if your implementation is correct by comparing your results to the references in the subfolder `output\testscenes`

Next time

- Ray-surface intersection
- Shading
- Reflection, refraction, shadows