Imatge Sintètica

Ray Tracing for Realistic Image Synthesis

Ricardo Marques
(ricardo.marques@upf.edu)

Group de Tecnologies Interactives (GTI)
Departament de Tecnologies de la Informació i les Comunicacions (DTIC)
Universitat Pompeu Fabra (UPF)

Edifici Tànger - Office 55.106

Framework - Ray, Camera, Sphere and EqSolver

2017/2018

Section 1

Ray

Ray - Definition

- Used to represent the concept of ray for ray tracing
  - Has an origin **o** and direction **d**
  - *minT* and *maxT* represent the ray beginning and ray end
  - *depth* is the depth of the ray (number of bounces)

```
class Ray
{
public:
    (...)

    // Ray public data
    Vector3D o;            // Ray origin
    Vector3D d;            // Ray direction
    mutable double minT;   //
    mutable double maxT;   //
    size_t depth;          // Ray depth (or number of bounces)
};
```

Ray - Constructors

- A *Ray* can be *constructed* in different ways:

```
// Constructors
Ray();
Ray(const Vector3D &ori, const Vector3D &dir,
    size_t dep = 0, double start = Epsilon,
    double end = INFINITY);
```

- By default, a *Ray* has value $\mathbf{d} = (0,0,0)$, $\mathbf{o} = (0,0,0)$, $depth = 0$, $minT = 0$ and $maxT = INFINITY$
- Note that there are **default parameters** in the second constructor
  - The expression *Ray(origin, direction)* is valid!
  - It can take 2 to 5 arguments
- A *Ray* can be written to the standard output

Section 2

Camera

Camera - Definition

- **Abstract class** used to represent *all* cameras
  - *cameraToWorld* allows transforming from camera coordinates to world coordinates
  - Has a *Film* containing the image
  - *aspect* contains the aspect ratio value (width/height)

```
class Camera
{
public:
    (...)
     /* General Camera data */
     // The cameraToWorld transformation matrix
     Matrix4x4 cameraToWorld;
     // Film to store and handle the actual image
     const Film &film;
     // Aspect ratio (based on the film size)
     double aspect;
};
```

Camera - Constructors

- A *Camera* can only be *constructed* by passing a reference to a *Matrix4x4* and a reference to a *Film*:

```
// Constructors

Camera() = delete;
Camera(const Matrix4x4 &cameraToWorld_,
       const Film &film_);
```

- The default constructor is explicitly disabled!
- A *Camera* **cannot** be written to the standard output (stream insertion operator '<<' not overloaded)

Camera - Others

- The implementation of the class *Camera* offers two useful methods:

```
// Returns a camera ray in WORLD COORDINATES
// which passes through (u, v)
virtual Ray generateRay(const double u,
                        const double v) const = 0;

// Convert from NDC to camera space
virtual Vector3D ndcToCameraSpace(const double u,
                        const double v) const = 0;
```

Section 3

EqSolver

EqSolver - Definition

- Class used to solve equations of second degree
- Resorts to an auxiliary structure called *rootValues*

```
struct rootValues
{
    unsigned int nValues;
    double values[2];
};
```

- *nValues*: used to store the number of solutions of the equation
- *values*: used to store the values of the solutions
- If *nValues*=0, then the values of *values* are meaningless

EqSolver - Constructor and Methods

- The class EqSolver has a single constructor

```
class EqSolver
{
public:
    EqSolver();

    (...)

};
```

- It offers a method to solve second degree equations

```
bool rootQuadEq(double c2, double c1,
                double c0, rootValues &res);
```

  - Returns true if there are solutions, and false otherwise

Section 4

Sphere

Sphere - Definition and Methods

- Class used to represent Spheres

```cpp
class Sphere : public Shape
{
public:
    Sphere() = delete;
    Sphere(const double radius_,
           const Matrix4x4 &t);

    virtual bool rayIntersectP(const Ray &ray) const;
    std::string toString() const;

private:
    // The center of the sphere in local
    // coordinates is (0, 0, 0).
    double radius;
};
```