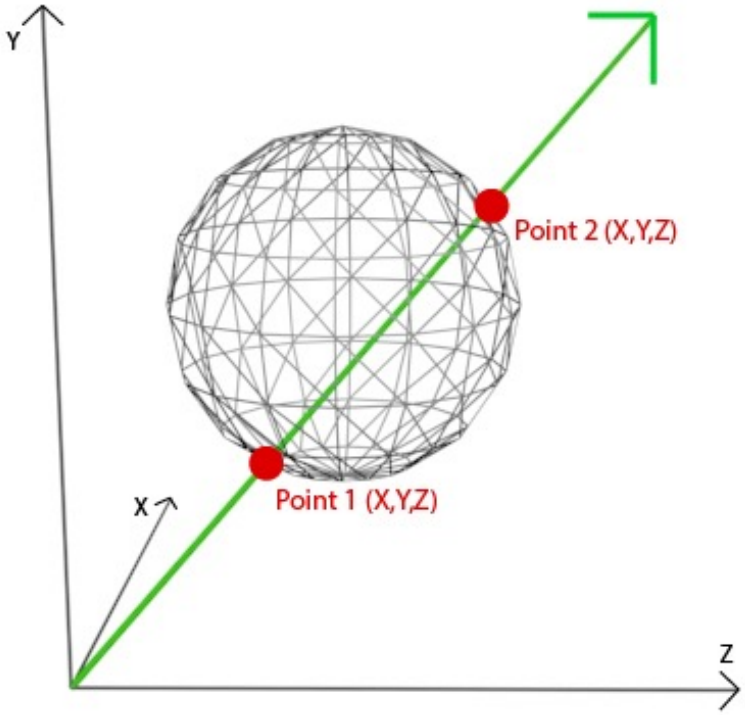
Mestrado em
Engenharia Informática

VI-RT Ray Mesh Intersection

Visualização e Iluminação

Luís Paulo Peixoto dos Santos

RAY MESH INTERSECTION



Primitive-Geometry

```
typedef struct Primitive {
    Geometry *g;
    int material_ndx;
} Primitive;
```

```
typedef struct Intersection {
public:
    Point p;
    Vector gn;  // geometric normal
    Vector sn;  // shading normal
    Vector wo;
    float depth;
    BRDF *f;
    int FaceID;}
```

```
class Geometry {
public:
    Geometry () {}
    ~Geometry () {}
    bool intersect (Ray r, Intersection *isect) {
        return false; }
    BB bb; };
```

The Mesh class

```
class Mesh: public Geometry {
private:
    bool TriangleIntersect (Ray r, Face f, Intersection *isect);
public:
    int numFaces;
    std::vector<Face> faces;
    int numVertices;
    std::vector<Point> vertices;
    int numNormals;
    std::vector<Vector> normals;
    bool intersect (Ray r, Intersection *isect);

    Mesh(): numFaces(0), numVertices(0), numNormals(0) {}
};
```

Face and Ray

```
typedef struct Face {
    int vert_ndx[3];            // indices to vector of vertices (in Mesh)
    Vector geoNormal;           // geometric normal
    bool hasShadingNormals;     // are there per vertex shading normals ??
    int vert_normals_ndx[3];    // indices to veritices normals
    BB bb;      // face bounding box
} Face;
```

```
class Ray {
public:
    Point o; // ray origin
    Vector dir; // ray direction
    Ray () {}
    Ray (Point o, Vector d): o(o),dir(d) {}
    ~Ray() {}
};
```

Mesh intersect

```
bool Mesh::intersect (Ray r, Intersection *isect) {
  bool intersect = false;
  Intersection min_isect, curr_isect;
  float min_depth=MAXFLOAT;

  // intersect the ray with the mesh BB
  if (!bb.intersect(r)) return false;

  for (auto face_it=faces.begin() ; face_it != faces.end() ; face_it++) {
    if (! TriangleIntersect(r, *face_it, &curr_isect)) continue;
        intersect = true;
        if (curr_isect.depth < min_depth) {  // this is closer
            min_depth = curr_isect.depth;
            min_isect = curr_isect;
  }     }
  return intersect;  }
```

AABB intersect

```
typedef struct BB {
    Point min, max;
    bool intersect (Ray r) { ... }
} BB;
```

For ray / axis aligned bounding box (AABB) intersection see:

- PBRT book, 3rd edition, sec 3.1.2, pags 125..128 + 214,217,221
  www.pbrt.org

- Shirley, P., Wald, I., Marrs, A. (2021).
  Ray Axis-Aligned Bounding Box Intersection.
  Ray Tracing Gems II. Apress, Berkeley, CA.
  https://doi.org/10.1007/978-1-4842-7185-8_2

Triangle intersect

```
bool Mesh::TriangleIntersect (Ray r, Face f, Intersection *isect) {

    if (!f.bb.intersect(r)) return false;

    ...
}
```

For ray / triangle intersection see:

- PBRT book, 3rd edition, sec 3.6.2, pags 157.. www.pbrt.org
- M¨oller, T., and B. Trumbore. 1997. Fast, minimum storage ray–triangle intersection. Journal of Graphics Tools 2(1), 21–28
  https://en.wikipedia.org/wiki/M%C3%B6ller%E2%80%93Trumbore_intersection_algorithm
- Woop, S., C. Benthin, and I. Wald. 2013. Watertight ray/triangle intersection. Journal of Computer Graphics Techniques (JCGT) 2(1), 65–82.

Intersection Information

```
typedef struct Intersection {
public:
    Point p;                        float depth;
    Vector gn;                      BRDF *f;
    Vector sn;                      int FaceID;
    Vector wo;    }
```

- `wo` is the light outgoing direction: `wo = -1.f * ray.dir`

- make sure the normal (`gn` and `sn`) points to the same side of the surface as `wo`
  There is a method in `class Vector` to help with this:

```
normal = normal.Faceforward(wo);
```

Intersection Information

```
typedef struct Intersection {
public:
    Point p;                          float depth;
    Vector gn;                        BRDF *f;
    Vector sn;                        int FaceID;
    Vector wo;    }
```