Mestrado em
Engenharia Informática

VI-RT Scene Loading

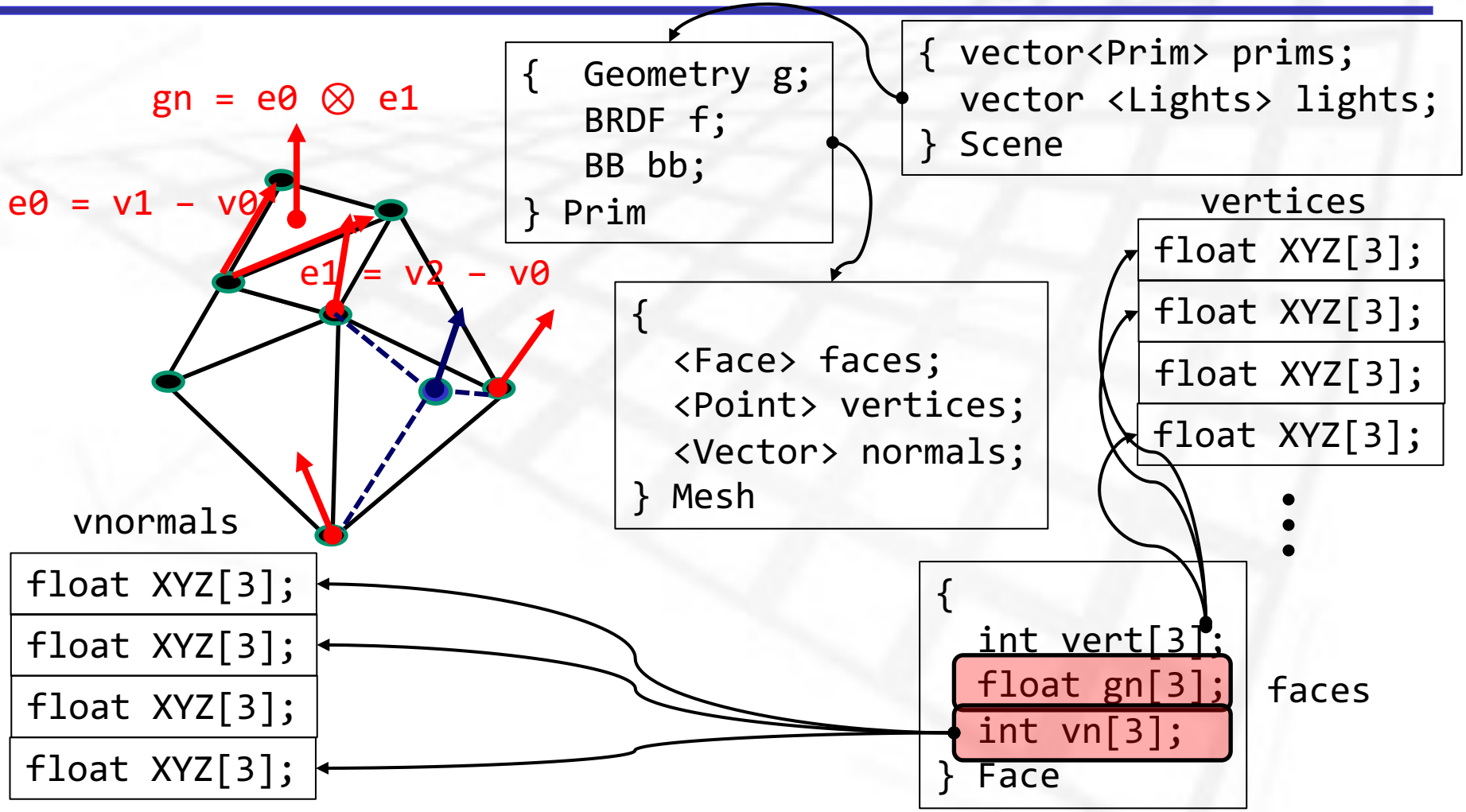
Visualização e Iluminação

Luís Paulo Peixoto dos Santos

SCENE LOADING

[Vecteezy.com]

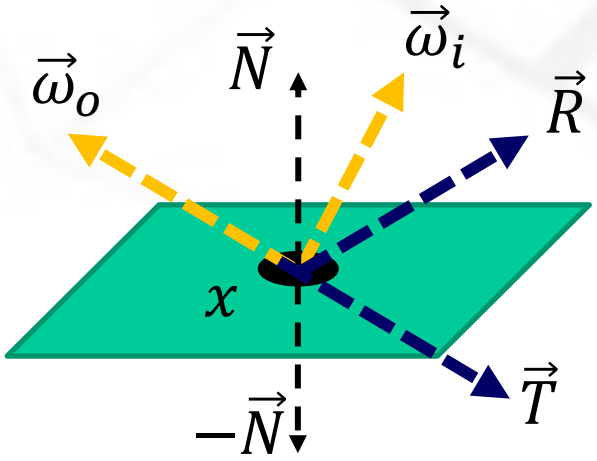
Mesh Representation



X-Phong Representation

$$L(\vec{\omega}_o \leftarrow x \leftarrow \vec{\omega}_i) = k_a * L_a +$$

$$+ L(x \leftarrow \vec{\omega}_i) * (k_d * (\vec{\omega}_i \cdot \vec{N}) + k_s * (\vec{\omega}_i \cdot \vec{R})^{N_s} + k_t * (\vec{\omega}_i \cdot \vec{T})^{N_s})$$




```
class Phong:: public BRDF {
    RGB Ka;
    RGB Kd;
    RGB Ks;
    float Ns;
    RGB Kt;
}
```

Wavefront .obj file

```
# List of geometric vertices, with x, y, z coordinates
v 0.123 0.234 0.345
v ... ...

# List of texture coordinates:(u, [v, w]) in [0 ... 1]
vt 0.500 [1 [0]]
vt ... ...

# List of vertex normals in (x,y,z) form
vn 0.707 0.000 0.707
vn ... ...
```

https://en.wikipedia.org/wiki/Wavefront_.obj_file

Wavefront .obj file

```
# which materials definition file to use
mtllib [external .mtl file name]

# which material to use in the subsequent objects
usemtl [material name]
```

https://en.wikipedia.org/wiki/Wavefront_.obj_file

Wavefront .obj file

```
# group subsequent faces onto an object
o [object name]

# Polygonal face element: f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 ...
# arguments are the vertices' indices according to their order in the file
f 1 2 3
f 3/1 4/2 5/3
f 6/4/1 3/5/3 7/6/5
f 6//3 2//1 7//2

# Note that for us, at this stage, an object is ALWAYS a MESH
# The faces following the 'o' command are the mesh triangles
```

https://en.wikipedia.org/wiki/Wavefront_.obj_file

Wavefront .mtl file

```
newmtl my_mtl
        Ka 0.0435 0.0435 0.0435
        Kd 0.1086 0.1086 0.1086
        Ks 0.0000 0.0000 0.0000
        Tf 0.9885 0.9885 0.9885
        Ns 10.0000
```

http://paulbourke.net/dataformats/mtl/

class Scene – load OBJ

```
class Scene {
    std::vector <Primitive *> prims;
    std::vector <BRDF *> BRDFs;
public:
    std::vector <Light *> lights;
    int numPrimitives, numLights, numBRDFs;


    Scene (): numPrimitives(0), numLights(0), numBRDFs(0) {}
    bool Load (const std::string &fname);
    ...
}
```

TINY OBJ LOADER – Reading OBJ

```
#define TINYOBJLOADER_IMPLEMENTATION
#include "tiny_obj_loader.h"
using namespace tinyobj;

bool Scene::Load (const std::string &fname) {
    ObjReader myObjReader;
    int FaceID = 0;

    ObjReader myObj;
    // this loader triangulates the faces
    if (!myObj.ParseFromFile(fname)) return false;
    ...
```

https://github.com/tinyobjloader/tinyobjloader

TINY OBJ LOADER – materials

```
...
const vector<material_t> materials = myObj.GetMaterials();
for (auto it = materials.begin() ; it != materials.end() ; it++) {
    Phong *mat = new Phong;
    // Ka
    mat->Ka.R = it->ambient[0];
    mat->Ka.G = it->ambient[1];
    mat->Ka.B = it->ambient[2];
    // Kd, Ns, Ks, Kt
    ...

    BRDFs.push_back (mat);
    numBRDFs++;
}
```

https://github.com/tinyobjloader/tinyobjloader

TINY OBJ LOADER – meshes and vertices

```
...
// access the vertices
const tinyobj::attrib_t attrib = myObj.GetAttrib();
float *vtcs = attrib.vertices;

// access the shapes (each shape is one mesh)
const vector<shape_t> shps = myObj.GetShapes();
```

https://github.com/tinyobjloader/tinyobjloader

TINY OBJ LOADER -mesh

```
// iterate over shapes (meshes)
for (auto shp = shps.begin() ; shp != shps.end() ; shp++) {
        Primitive *p = new Primitive;
        Mesh *m = new Mesh;
        p->g = m;
        // assume all faces in the mesh have the same material
        p->material_ndx = shp->mesh.material_ids[0];

       // the primitive's geometry bounding box is computed on the fly
        // initially set BB.min and BB.max to be the first vertex
        const int V1st = shp->mesh.indices.begin()->vertex_index * 3;
        m->bb.min.set(vtcs[V1st], vtcs[V1st+1], vtcs[V1st+2]);
        m->bb.max.set(vtcs[V1st], vtcs[V1st+1], vtcs[V1st+2]);

       ...
```

https://github.com/tinyobjloader/tinyobjloader

TINY OBJ LOADER - mesh

```
...
// add faces and vertices
std::set<rehash> vert_rehash;
for (auto v_it=shp->mesh.indices.begin(); v_it!=shp-> mesh.indices.end() ;) {
   Face *f = new Face;
   Point myVtcs[3];
   // process 3 vertices
   for (int v=0 ; v<3 ; v++) {
     const int objNdx = v_it->vertex_index;
     myVtcs[v].set(vtcs[objNdx*3], vtcs[objNdx*3+1], vtcs[objNdx*3+2]);
     if (v==0) {
         f->bb.min.set(myVtcs[0].X, myVtcs[0].Y, myVtcs[0].Z);
         f->bb.max.set (myVtcs[0].X, myVtcs[0].Y, myVtcs[0].Z);
     }  else face->bb.update(myVtcs[v]);
     ...  next: add face and vertex to mesh
```

https://github.com/tinyobjloader/tinyobjloader

TINY OBJ LOADER - mesh

```
// add faces and vertices
std::set<rehash> vert_rehash;
for (auto v_it=shp->mesh.indices.begin(); v_it!=shp-> mesh.indices.end() ;) {
   ...
   for (int v=0 ; v<3 ; v++) {
     ...
     // add vertex to mesh if new
     rehash new_vert={objNdx, 0};
     auto known_vert = vert_rehash.find(new_vert);
     if (known_vert == vert_rehash.end()) { // new vertice, add it to the mesh
         new_vert.ourNdx = m->numVertices;
         vert_rehash.insert(new_vert);
         m->vertices.push_back(myVtcs[v]);     m->numVertices++;
         // register in the face
         f->vert_ndx[v] = new_vert.ourNdx;     m->bb.update(myVtcs[v]);
     } else f->vert_ndx[v] = known_vert->ourNdx;
     v_it++;  // next vértice within this face (there are 3)
   }    //    end vertices
   ...  next add face to mesh
```

TINY OBJ LOADER - mesh

```
    // add faces and vertices
    std::set<rehash> vert_rehash;
    for (auto v_it=shp->mesh.indices.begin(); v_it!=shp-> mesh.indices.end() ;) {
       ...
       //  add face to mesh: compute the geometric normal
       Vector v1 = myVtcs[0].vec2point(myVtcs[1]);
       Vector v2 = myVtcs[0].vec2point(myVtcs[2]);
       f->edge1 = v1;  f->edge2 = v2;
       Vector normal = v1.cross(v2);
       f->geoNormal.set(normal.normalize());
       f->FaceID = FaceID++;
       // add face to mesh
       m->faces.push_back(*f);        m->numFaces++;
    }    // end iterate vértices in the mesh (shape)
    // add primitive to scene
    prims.push_back(p); numPrimitives++;
  }  // end iterate over shapes
  return true;
}
```