CPSC 314
Computer Graphics

Dinesh K. Pai

glTF part 2, Environment Maps

NOTICE:
Recordings of the lecture are provided to students enrolled in the course for self-study only. Any other use, including reproduction and sharing of links to materials, is strictly prohibited.

Preliminaries

- Today
  - glTF continued
  - Environment maps

2

glTF

- gl Transmission Format
- Introduced by Khronos – Industry consortium supporting OpenGL, WebGL, etc.
- Version 1.0 (2015) had some limitations
- Version 2.0 (2017) is much better, and is getting a lot of traction
  - Lot of industry support
  - I encourage you to start moving to it

- See https://www.khronos.org/gltf/ for resources
- The following are slides from Khronos's presentations https://www.khronos.org/assets/uploads/developers/library/2017-web3d/glTF-2.0-Launch_Jun17.pdf

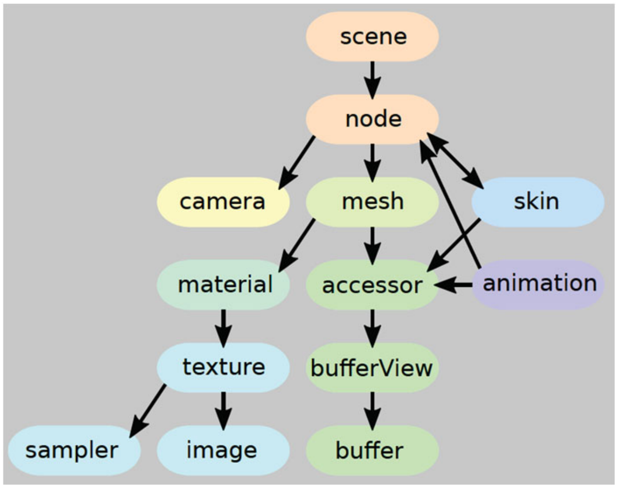
(Switch to pdf)

Support for glTF

- Microsoft PowerPoint (demo)
- Facebook
- Three.js can load and export https://threejs.org/examples/#webgl_loader_gltf
- VSCode (demo and file structure)

glTF structure

- A high level introduction
- Stores scene graph + textures + animation assets in a JSON file
- You now most of the pieces to understand it
- Very nice reference: https://www.khronos.org/files/gltf20-reference-guide.pdf



Environment maps

Steps for Texture Mapping

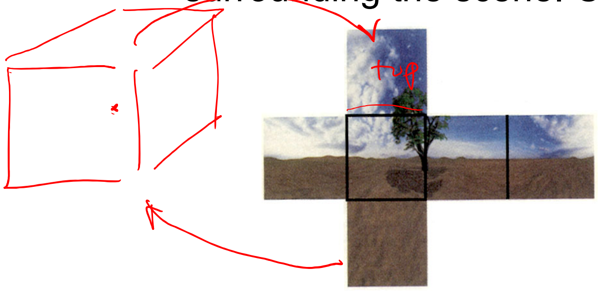
1. Create a *texture object* and load texels into it
2. Include *texture coordinates* with your vertices
3. Associate a *texture sampler* with each texture map used in shader
4. Retrieve texel values

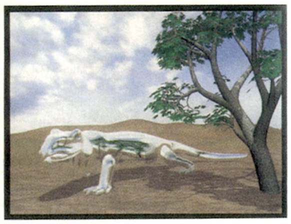
Generating your own Texture Coordinates

- Can be done in Maya, 3DS Max, Blender and other 3D modeling software. This is how it's done in production applications
- Legacy OpenGL had a function (glTexGen) to do this, removed from current versions
- In production, coordinates are designed with model (or "painted" on 3D model)
- Useful texture coordinates can often be computed in shaders (e.g., projection, environment maps)

Environment cube maps

- Textures can also be used to model the environment in the distance around the object being rendered.
- In this case, we typically use 6 square textures representing the faces of a large cube surrounding the scene. Called a "Sky Box"

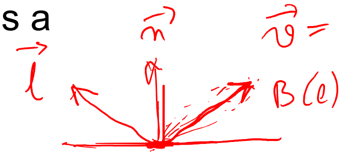


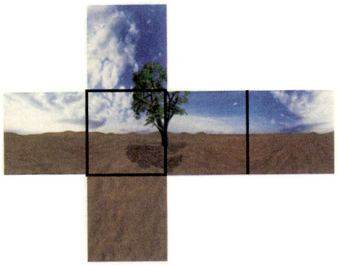


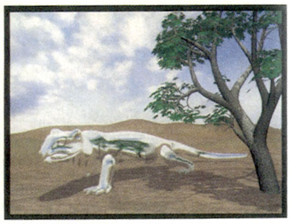
10

Environment cube maps

- Each texture pixel represents the color as seen along one direction in the environment.
- This is called a *cube map*. GLSL provides a cube-texture data type, samplerCube, specifically for this purpose.

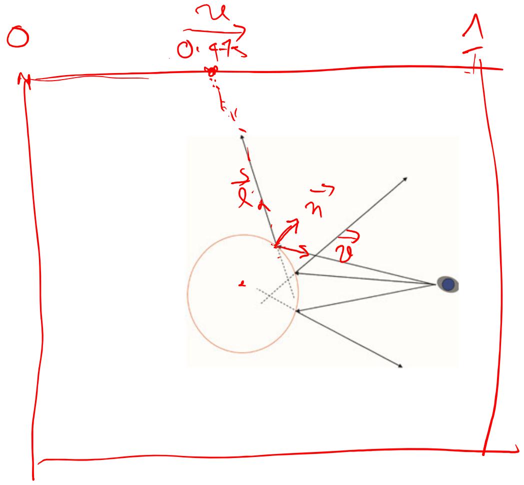






11

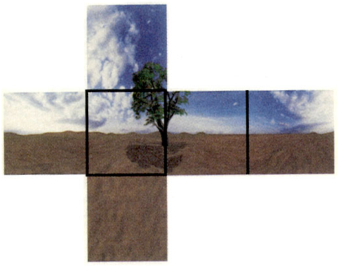
Geometry of Cube Mapping

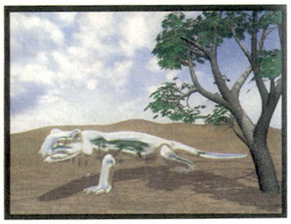


12

Environment cube maps

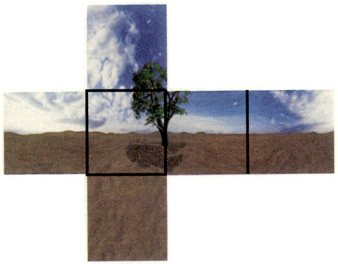
- During the shading of a point, we can treat the material at that point as a perfect mirror and fetch the environment data from the appropriate incoming direction.

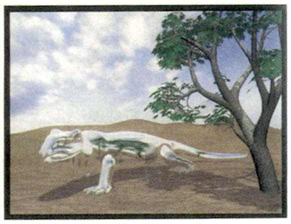




Environment map shader

- We calculated $B(\vec{v})$ in a previous lecture.
- This bounced vector will point points towards the environment direction, which would be observed in a mirrored surface.
- By looking up the cube map, using this direction, we give the surface the appearance of a mirror.





Environment map shader

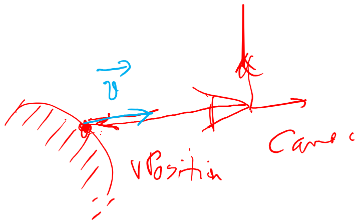
- Fragment shader

```
#version 330
uniform samplerCube uTexUnit0;
in vec3 vNormal;
in vec4 vPosition;
out vec4 fragColor;

vec3 reflect(vec3 w, vec3 n){
   return n*(dot(w,n)*2.0) - w; // bounce vector
}

void main() {
   vec3 normal = normalize(vNormal);
   vec3 reflected = reflect(normalize(vec3(-vPosition)), normal);
   vec4 texColor0 = samplerCube(uTexUnit0, reflected);
   fragColor = vec4(texColor0.r, texColor0.g, texColor0.b, 1.0);;
}
```

Can use builtin



Environment map shader

- -vPosition represents the view vector $\vec{v}$
- samplerCube is a special GLSL function that takes a direction vector and returns the color stored at this direction in the cube texture map.
- Here we assume eye-coordinates, but frame changes may be needed.