174A
TA: FranklinFang
Discuss Section
11/7/2014

Some keywords

- Aliasing and anti-aliasing
- Texture sampling/filtering
- Mipmapping
- Anisotropic filtering

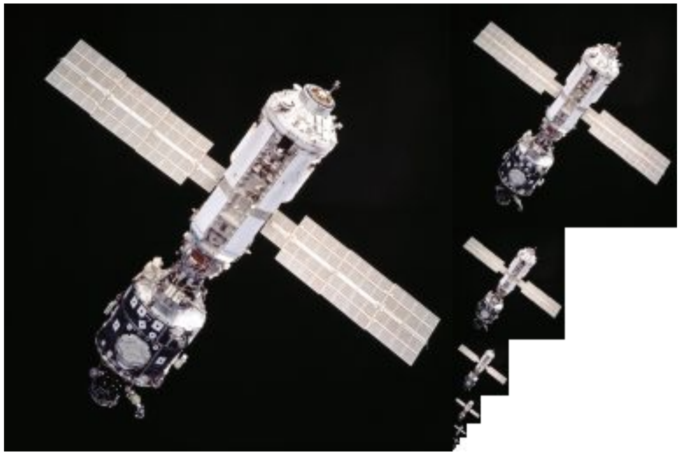
Different sampling methods

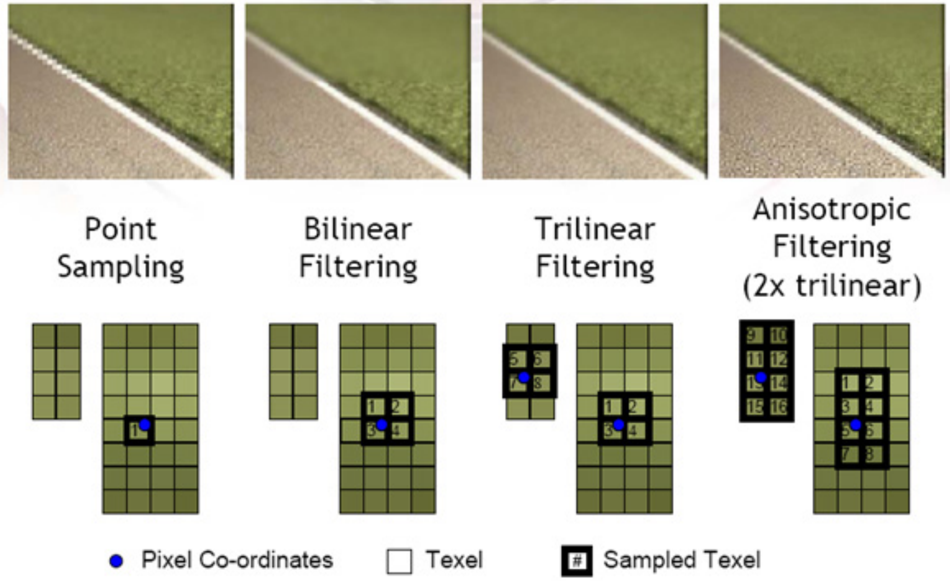
- simplest: nearest neighbor. Serious aliasing artifact.

- mipmaps are pre-calculated optimzied collections of images that accompany a main texture. intended to increase rendering speed and reduce aliasing artifacts.

- nearest-neighbor with mipmapping. Better, reduce some aliasing. Jump when switching to a different level of mipmap.

mipmaps

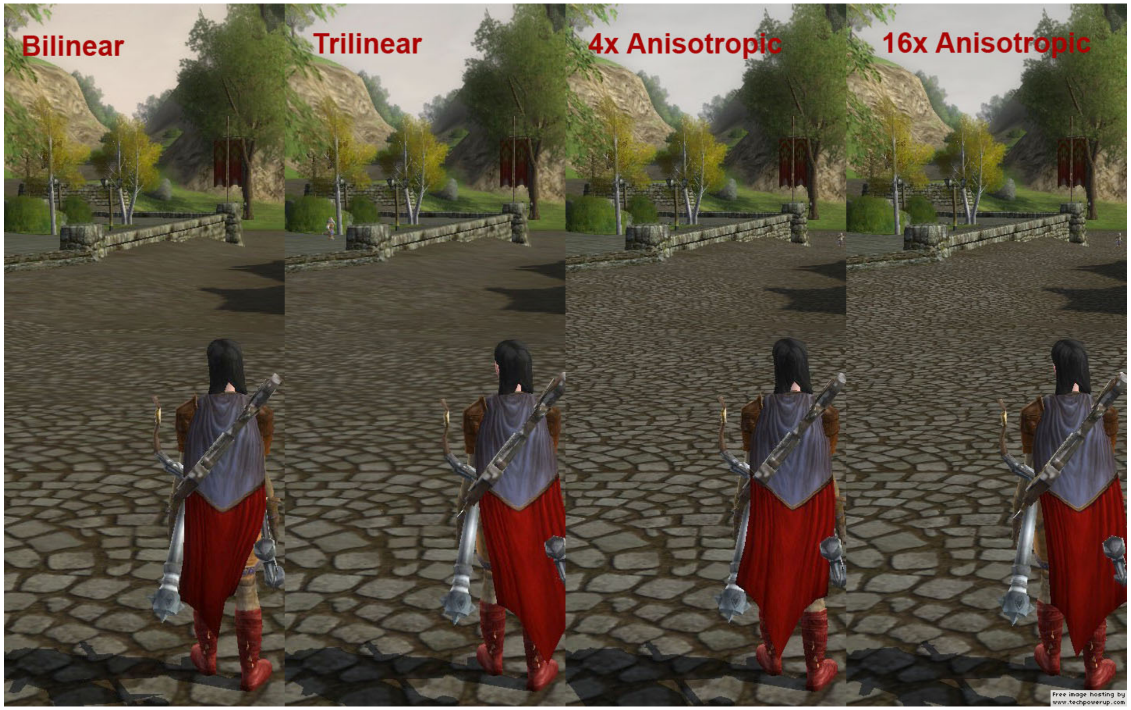




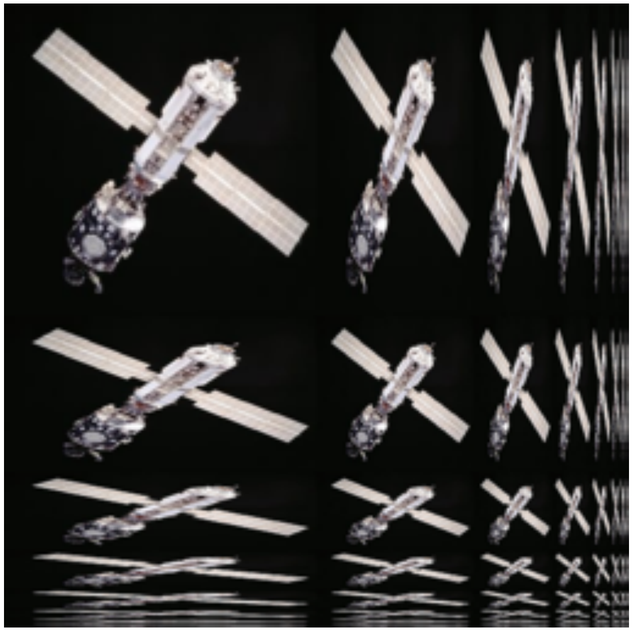
- Bilinear filtering. Use the four nearest texels and combine them by weighted average according to the distance of the sampling position to the four texels. Problem: jump of qulaity when switching from one mipmap to the other

- Trilinear filtering. do bilinear filtering on two closest mipmap, then linear interpolate between two.

- Anisotropic filtering: for a surface with oblique viewing angle, sample from a pre-calculated ripmap. Anisotropic filtering retains the "sharpness" of a texture normally lost by MIP map texture's attempts to avoid aliasing.



ripmaps



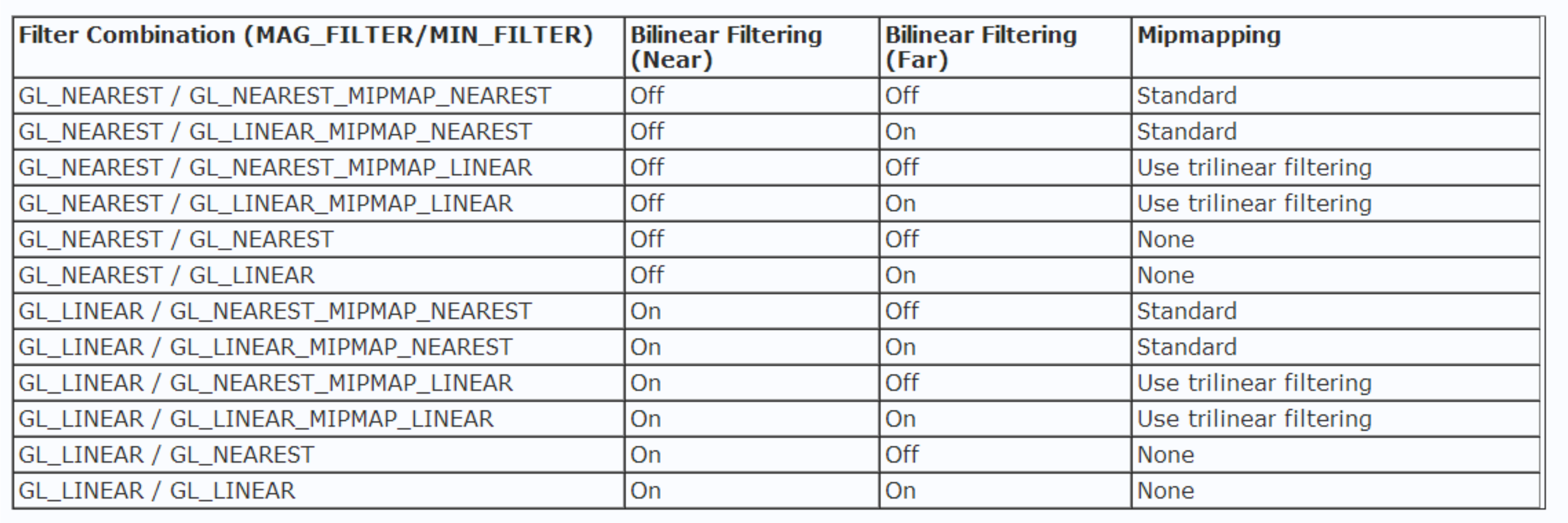
Texture sampling in OpenGL

- glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, <texture shrinkage filter>);
- glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, <texture expansion filter>);

- The filter value set for GL_TEXTURE_MIN_FILTER is used whenever a surface is rendered with smaller dimensions than its corresponding texture bitmap (far away objects). Whereas the filter value for GL_TEXTURE_MAG_FILTER is used in the exact opposite case – a surface is bigger than the texture being applied (near objects).

- GL_TEXTURE_MIN_FILTER, parameter three can be:
  - GL_NEAREST_MIPMAP_NEAREST
  - GL_LINEAR_MIPMAP_NEAREST
  - GL_NEAREST_MIPMAP_LINEAR
  - GL_LINEAR_MIPMAP_LINEAR
  - GL_NEAREST
  - GL_LINEAR
- GL_TEXTURE_MAG_FILTER, parameter three can be:
  - GL_LINEAR
  - GL_NEAREST

Texture Sampling in OpenGl



Filter Combination (MAG_FILTER/MIN_FILTER)	Bilinear Filtering (Near)	Bilinear Filtering (Far)	Mipmapping
GL_NEAREST / GL_NEAREST_MIPMAP_NEAREST	Off	Off	Standard
GL_NEAREST / GL_LINEAR_MIPMAP_NEAREST	Off	On	Standard
GL_NEAREST / GL_NEAREST_MIPMAP_LINEAR	Off	Off	Use trilinear filtering
GL_NEAREST / GL_LINEAR_MIPMAP_LINEAR	Off	On	Use trilinear filtering
GL_NEAREST / GL_NEAREST	Off	Off	None
GL_NEAREST / GL_LINEAR	Off	On	None
GL_LINEAR / GL_NEAREST_MIPMAP_NEAREST	On	Off	Standard
GL_LINEAR / GL_LINEAR_MIPMAP_NEAREST	On	On	Standard
GL_LINEAR / GL_NEAREST_MIPMAP_LINEAR	On	Off	Use trilinear filtering
GL_LINEAR / GL_LINEAR_MIPMAP_LINEAR	On	On	Use trilinear filtering
GL_LINEAR / GL_NEAREST	On	Off	None
GL_LINEAR / GL_LINEAR	On	On	None

Worst, Nearest:

```
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
```

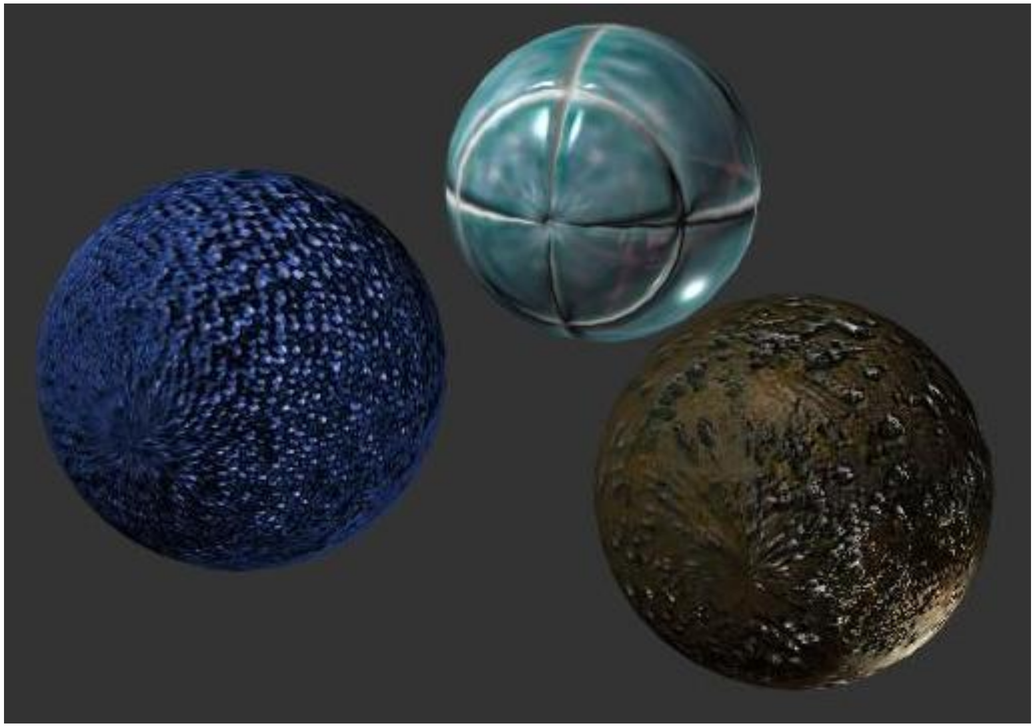
Best, Trilinear:

```
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR );
```

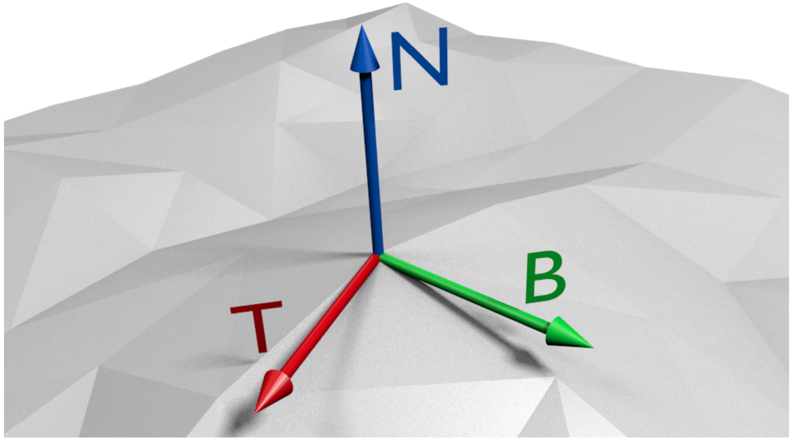
Trilinear vs nearest:
http://mrdoob.github.io/three.js/examples/webgl_materials_texture_filters.html

Bump-mapping: What is it:

http://mrdoob.github.com/three.js/examples/webgl_materials_bumpmap.html



- bump-mapping: typically done by perturbating surface normals of vertices.
- Achieved through shader by reading from a texture for surface normal, but the normal vectors stored in the normal map are expressed in the **tangent space** of each vertex.
- In that tangent space, normal of a vertex is originally (0,0,1), now use what is read from the normal texture map
- Need the representation of {T,B,N} in NDCS to transform the normal from tangent space CS to NDCS .



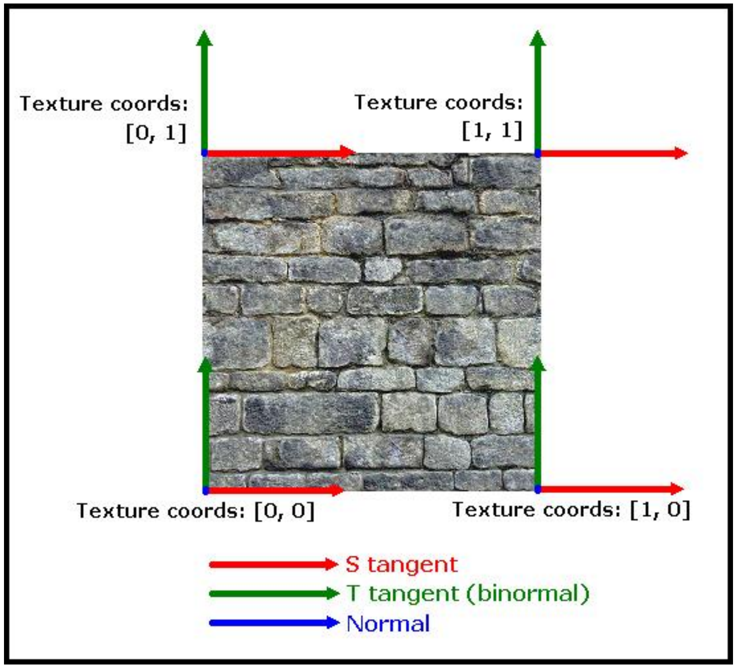
In tangent space:

tangent vector T = {1.0, 0.0, 0.0}

bitangent vector B = {0.0, 1.0, 0.0}

normal vector N = {0.0, 0.0, 1.0}

Calculating the tangent space CS in NDCS



```
generateNormalAndTangentPerFace
(float3 v1, float3 v2, text2 st1, text2 st2)
{
float3 normal = v1.crossProduct(v2);
 float coef = 1/ (st1.u * st2.v - st2.u * st1.v);
 float3 tangent;
 tangent.x = coef * ((v1.x * st2.v) + (v2.x * -st1.v));
tangent.y = coef * ((v1.y * st2.v) + (v2.y * -st1.v));
tangent.z = coef * ((v1.z * st2.v) + (v2.z * -st1.v));
 float3 bitangent = normal.crossProduct(tangent);
}
```

Just like normals: tangents and bitangents are accumulated for each faces connected to this vertex and then averaged via normalization.

More read here: http://www.terathon.com/code/tangent.html

Normal phong shading

- Either transform the L,V,H vectors to the tangent space, or transform the Normal to the NDCS.

Vertex shader

```
varying vec3 lightVec;
varying vec3 eyeVec;
varying vec2 texCoord;
attribute vec3 vTangent;
void main(void) {
        gl_Position = ftransform();
        texCoord = gl_MultiTexCoord0.xy;
        vec3 n = normalize(gl_NormalMatrix * gl_Normal);
        vec3 t = normalize(gl_NormalMatrix * vTangent);
        vec3 b = cross(n, t);
        vec3 vVertex = vec3(gl_ModelViewMatrix * gl_Vertex);
        vec3 tmpVec = gl_LightSource[0].position.xyz - vVertex;
        lightVec.x = dot(tmpVec, t);
        lightVec.y = dot(tmpVec, b);
        lightVec.z = dot(tmpVec, n);
        tmpVec = -vVertex;
        eyeVec.x = dot(tmpVec, t);
        eyeVec.y = dot(tmpVec, b);
        eyeVec.z = dot(tmpVec, n);
}
```

http://www.ozone3d.net/tutorials/bump_mapping_p4.php

Fragment shader

```
varying vec3 lightVec;
varying vec3 eyeVec;
varying vec2 texCoord;
uniform sampler2D colorMap;
uniform sampler2D normalMap;
uniform float invRadius;
void main (void) {
                float distSqr = dot(lightVec, lightVec);
                 float att = clamp(1.0 - invRadius * sqrt(distSqr), 0.0, 1.0);
                vec3 lVec = lightVec * inversesqrt(distSqr);
                vec3 vVec = normalize(eyeVec);
                 vec4 base = texture2D(colorMap, texCoord);
                 vec3 bump = normalize( texture2D(normalMap, texCoord).xyz * 2.0 - 1.0);
                vec4 vAmbient = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;
                float diffuse = max( dot(lVec, bump), 0.0 );
                vec4 vDiffuse = gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse * diffuse;
                float specular = pow(clamp(dot(reflect(-lVec, bump), vVec), 0.0, 1.0),
                gl_FrontMaterial.shininess );
                vec4 vSpecular = gl_LightSource[0].specular * gl_FrontMaterial.specular * specular;
                 gl_FragColor = ( vAmbient*base + vDiffuse*base + vSpecular) * att;
}
```

http://www.ozone3d.net/tutorials/bump_mapping_p4.php