

# **5. Computergraphik Übung: Texturing**

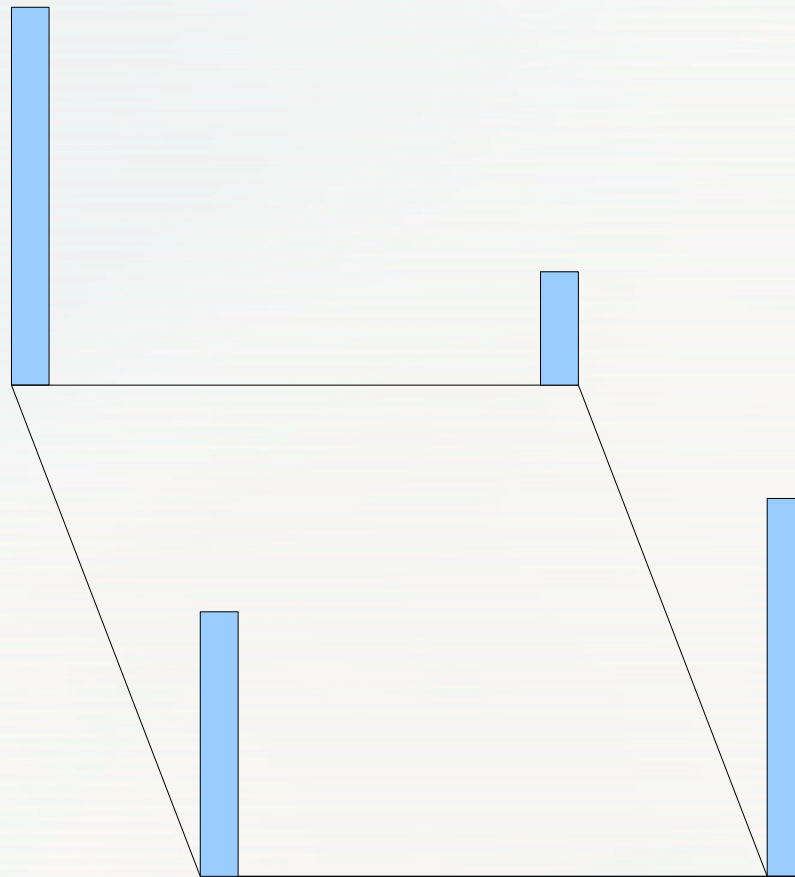
**Heni Ben Amor**

**Arbeitsgruppe Virtuelle Realität und Multimedia**

**TU Bergakademie Freiberg**

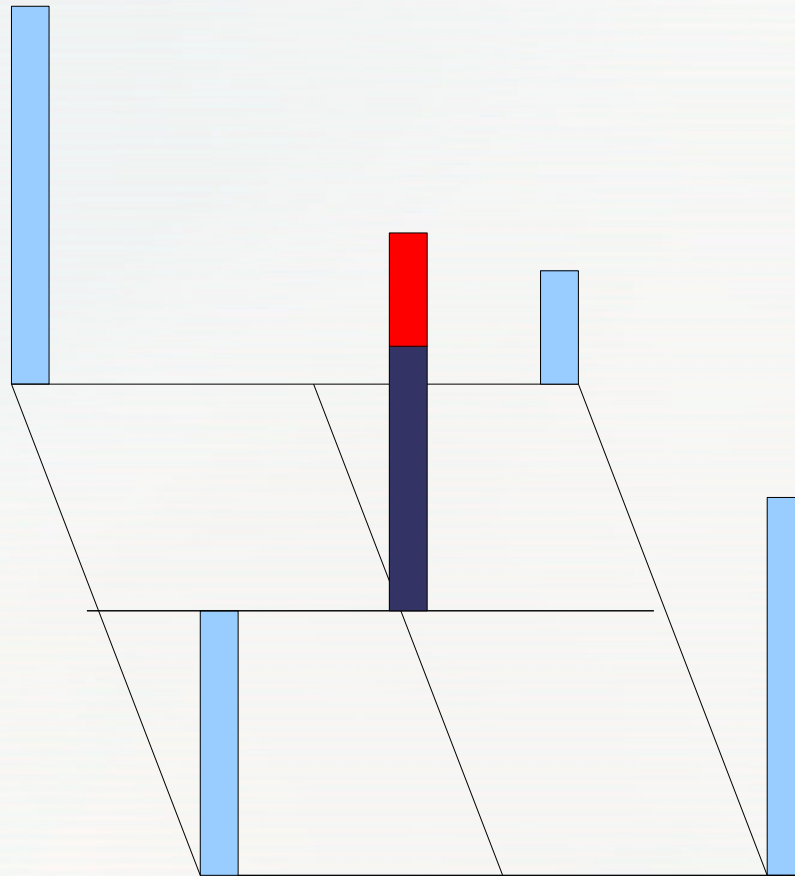
# Aufgabe 1: Fraktales Terrain

Startzustand: 4 Zufällige Eckpunkthöhen



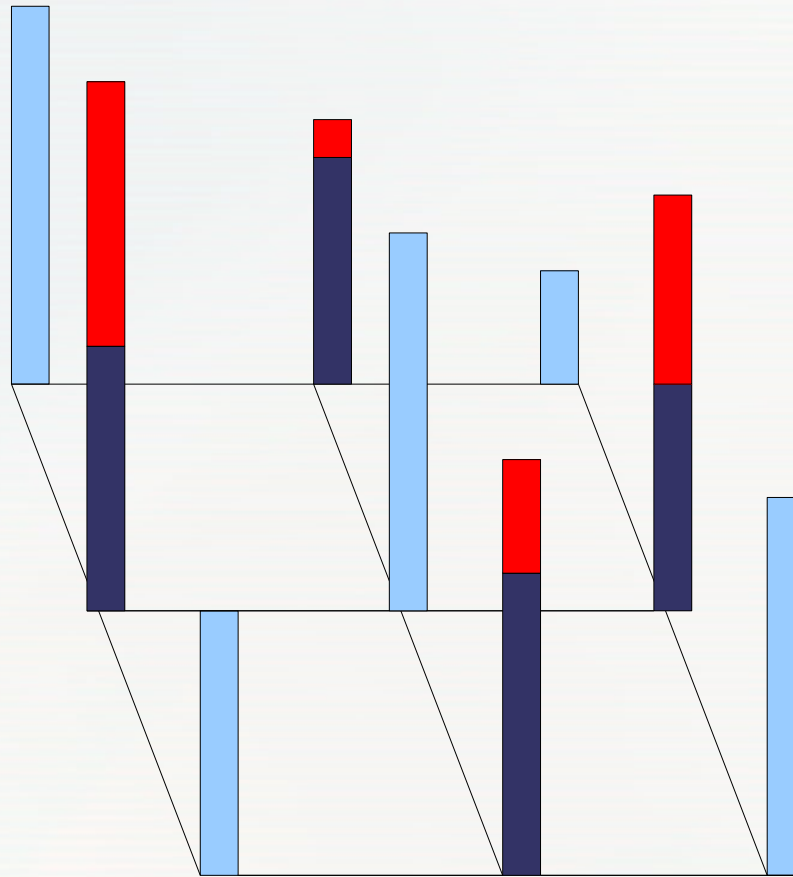
# Aufgabe 1: Fraktales Terrain

1. Iteration: Höhe des Mittelpunktes bestimmen  
Höhe = Mittelwert der 4 Eckpunkthöhen + Zufallszahl



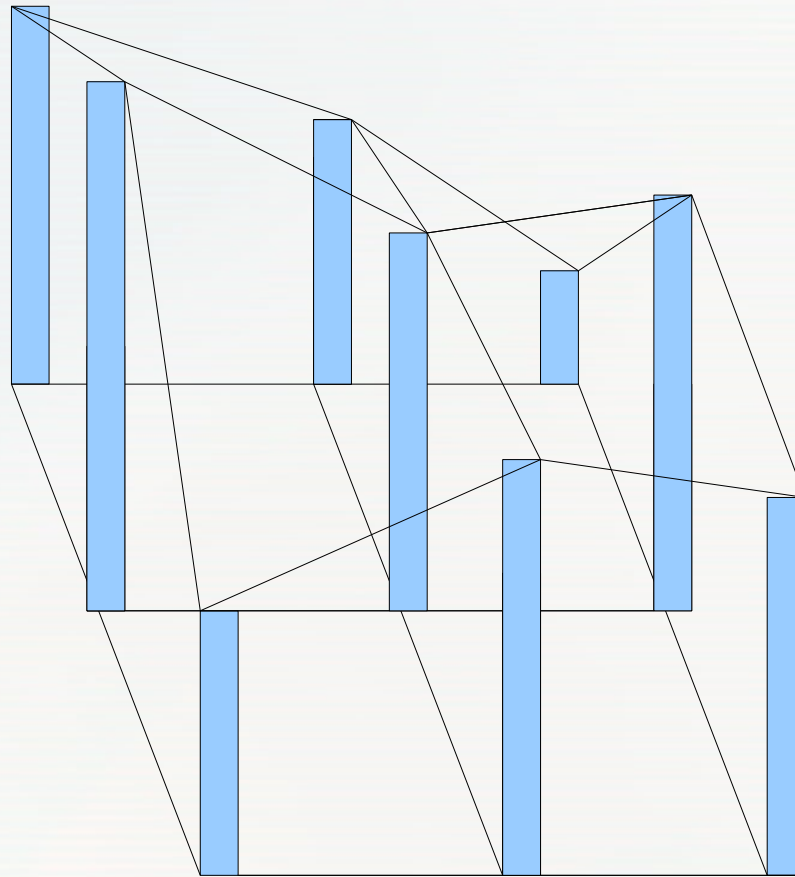
# Aufgabe 1: Fraktales Terrain

1. Iteration: Mittelwerte der Höhen der beiden Punkte der Kante + Zufallszahl



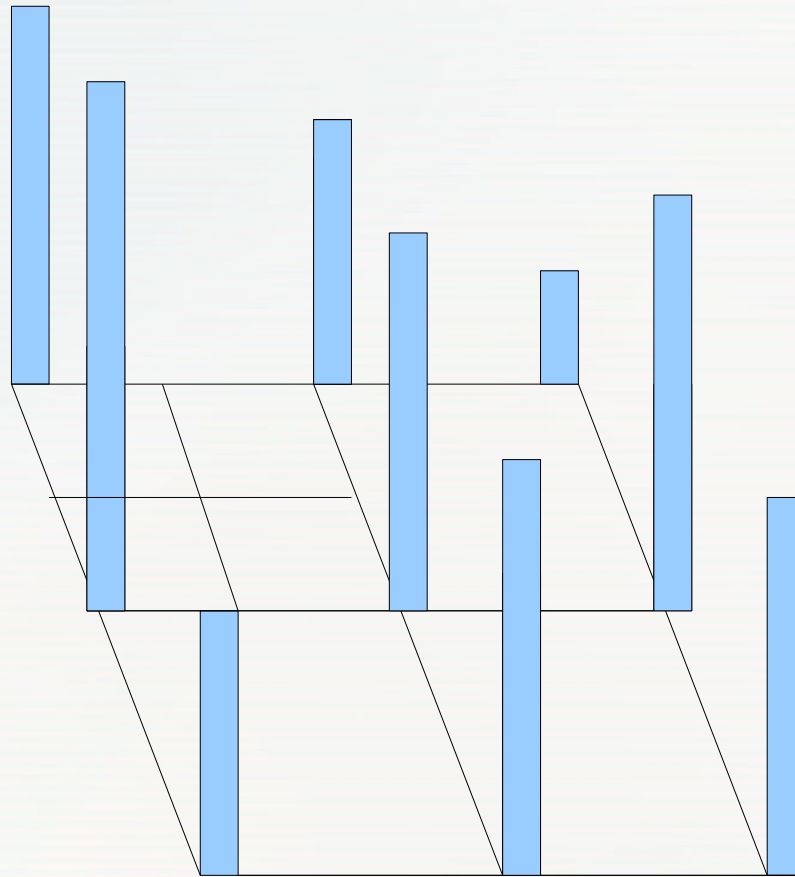
# Aufgabe 1: Fraktales Terrain

1. Iteration: Mittelwerte der Höhen der beiden Punkte der Kante + Zufallszahl



# Aufgabe 1: Fraktales Terrain

2. Iteration:



# Aufgabe 1: Fraktales Terrain

- **Was zu beachten ist:**
  - Anzahl der Vertices in jeder Dimension muss ungerade sein
  - Anzahl der Vertices muss  $2^n$  sein
  - Der Bereich der Zufallszahlen sollte der Funktion als Parameter übergeben werden
  - Der Bereich sollte bei jedem aufruf der Funktion durch 2 geteilt werden:

```
diamond_square(startx, starty, endx, endy, randMin/2.0, randMax/2.0)
```

# Aufgabe 2: Berghaus

```
osg::Vec4Array* colorArray = new osg::vec4Array;

colorArray->push_back(osg::Vec4(1.0f, 1.0f, 0.0f, 1.0f) );
colorArray->push_back(osg::Vec4(0.0f, 0.0f, 0.0f, 1.0f) );

osg::TemplateIndexArray<unsigned int,
                        osg::Array::UIntArrayType, 8, 8> *colorIndex;

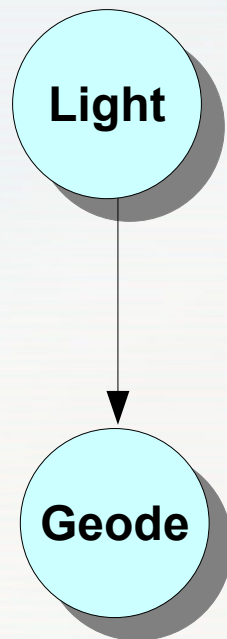
colorIndex = new osg::TemplateIndexArray<unsigned int, ...>

colorIndex->push_back(1);
colorIndex->push_back(0);

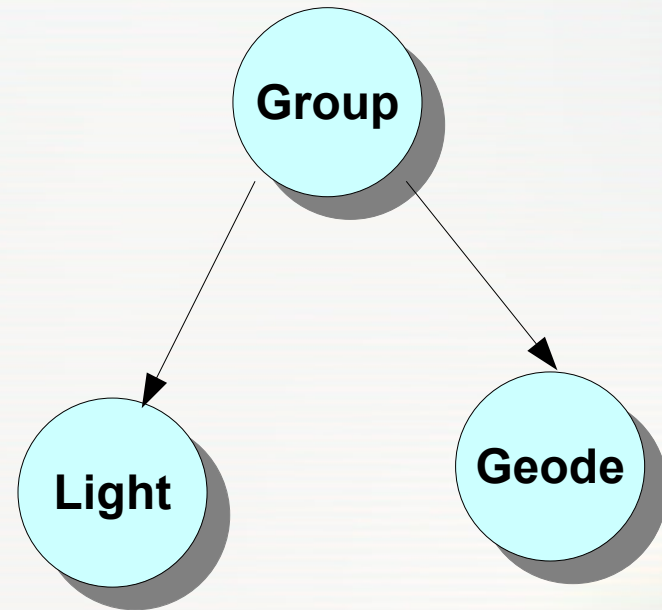
pyramidGeometry->setColorArray(colorArray);
pyramidGeometry->setColorIndices(colorIndex);
pyramidGeometry->setColorBinding(osg::Geometry::BIND_PER_VERTEX);
```

# Aufgabe 3: Lichter

- **Wichtig:**
  - Alle beleuchteten Objekte müssen im Szenengraphen unter dem betreffenden Licht hängen



oder



# Aufgabe 4: Pong vs. Gouraud



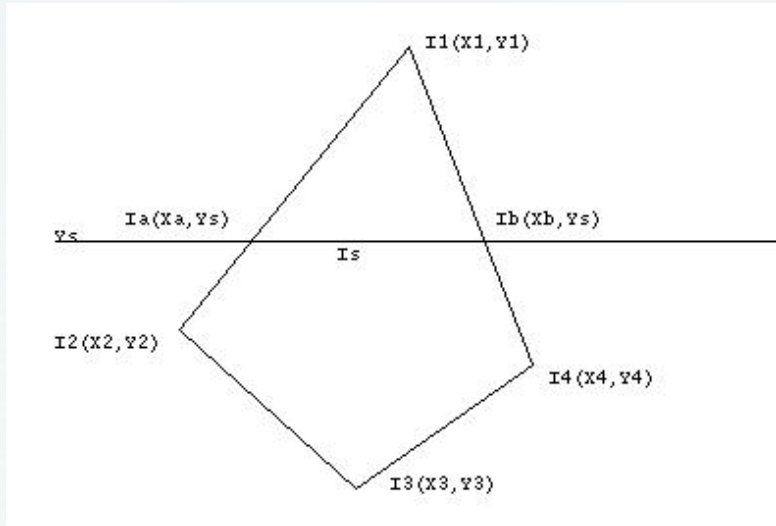
Gouraud Shading



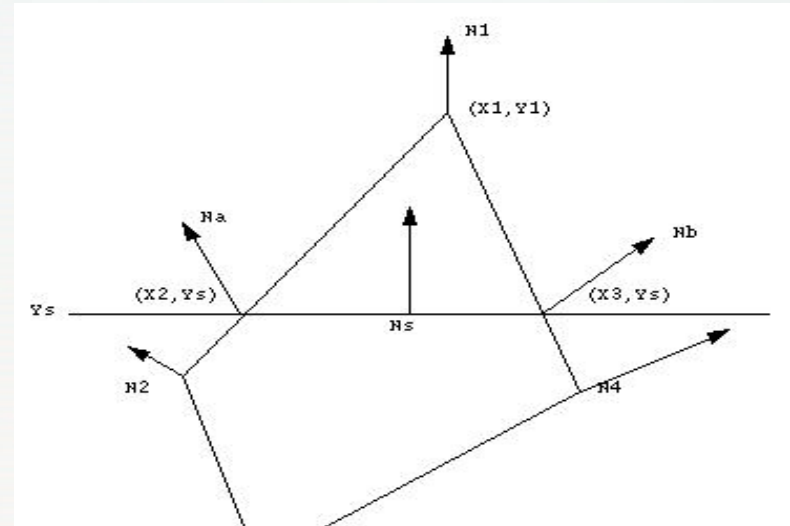
Phong Shading

- Bei beiden Verfahren wird interpoliert
- Unterschied: „Was wird interpoliert?“
- Bei Gouraud sind es Farbwerte der Eckpunkte
- Bei Phong sind es Normalenvektoren der Eckpunkte

# Aufgabe 4: Phong vs. Gouraud



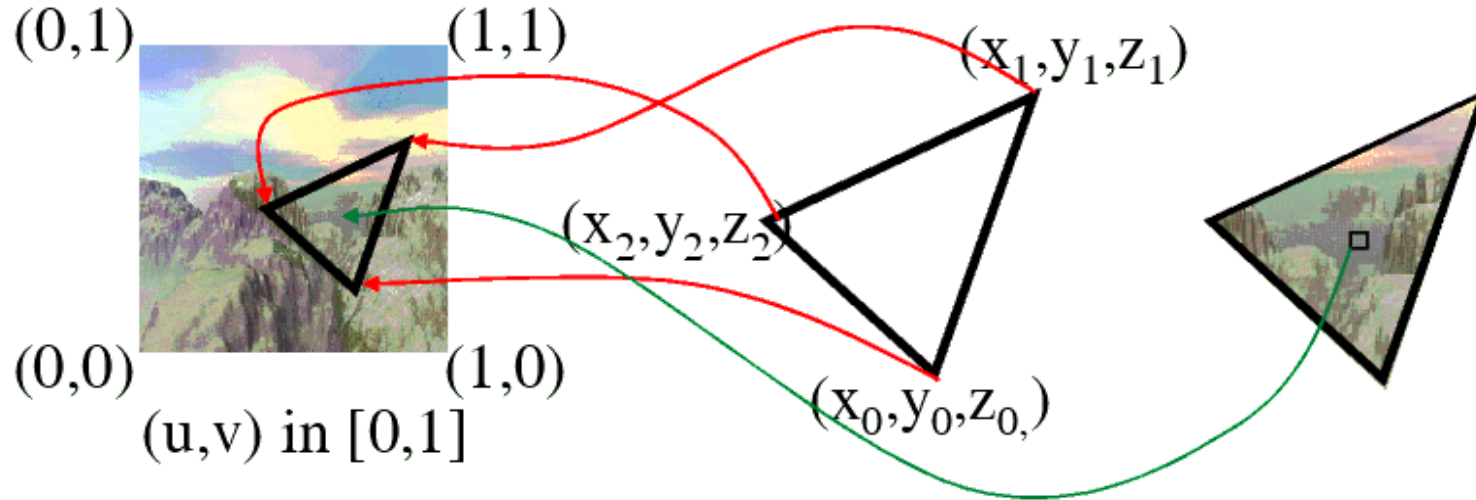
Gouraud



Phong

# Texturing

## XYZ $\rightarrow$ UV Parameter Space



- Projecting geometry to uv-space
  - per pixel
    - better quality; during rasterization; non-interactive renderers
  - per vertex
    - real-time applications
    - projector functions usually applied at modelling stage, results stored at vertices
    - when Gouraud shading, texture coordinates are not linearly interpolated
    - instead apply perspective correction!

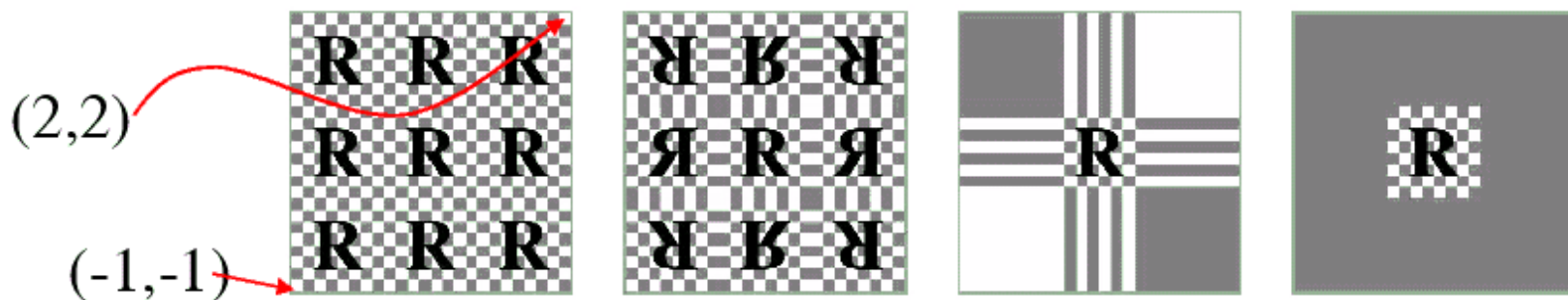
# Texturing

```
// array für u,v – Koordinaten erstellen  
osg::Vec2Array* texcoords = new osg::Vec2Array(4);  
  
// u,v Koordinaten setzen  
(*texcoords)[0].set(0.0, 1.0f)  
(*texcoords)[1].set(0.0, 0.0f)  
(*texcoords)[2].set(1.0, 0.0f)  
(*texcoords)[3].set(1.0, 1.0f)  
  
// u,v koordinaten übergeben  
geometry->setTexCoordArray(0, texcoords);
```

# Texturing

## Parameter Space $\rightarrow$ Texture Coordinates

- Corresponder functions (mapping  $uv \rightarrow$  texels)
  - What if  $(u,v) > 1.0$  or  $< 0.0$  ?
  - To repeat textures, use just the fractional part
    - Example:  $5.3 \rightarrow 0.3$
  - Repeat, mirror, clamp, border:



# Texturing

```
texture->setWrap(WrapParameter;WrapMode);
```

WrapParameter:

```
WRAP_S  
WRAP_T  
WRAP_R
```

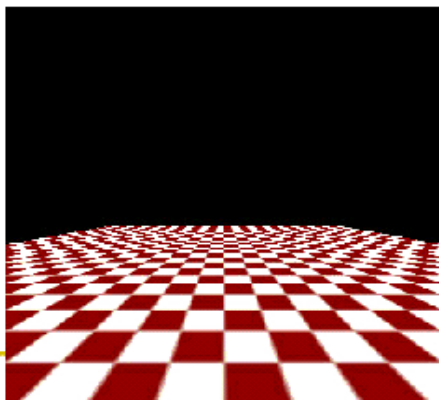
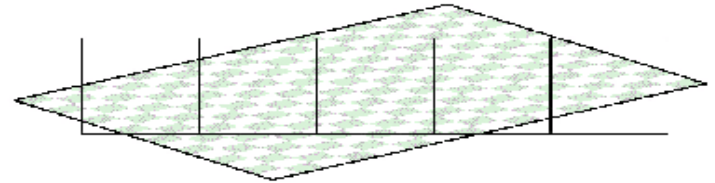
WrapMode:

```
CLAMP  
CLAMP_TO_EDGE  
CLAMP_TO_BORDER  
REPEAT  
MIRROR
```

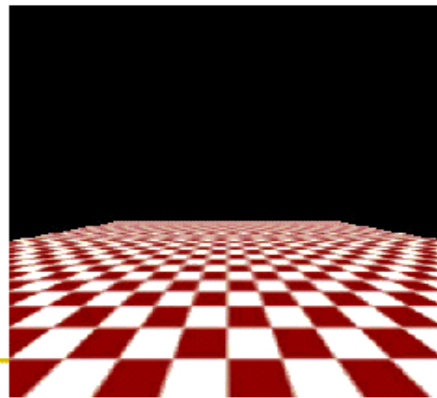
# Texturing

## Texture Minification

- one pixel covering many texels
- methods
  - select center texel? ("nearest neighbor")
    - → aliasing problems
  - average colors of all texels covered by pixel?
    - → can be expensive (not constant time)
  - mip-mapping ...



no mipmapping



mipmapping

[http://www.flipcode.com/articles/article\\_advgltextures.shtml](http://www.flipcode.com/articles/article_advgltextures.shtml)

# Texturing

```
texture->setFilter(FilterParameter;FilterMode);
```

FilterParameter:

```
MIN_FILTER  
MAG_FILTER
```

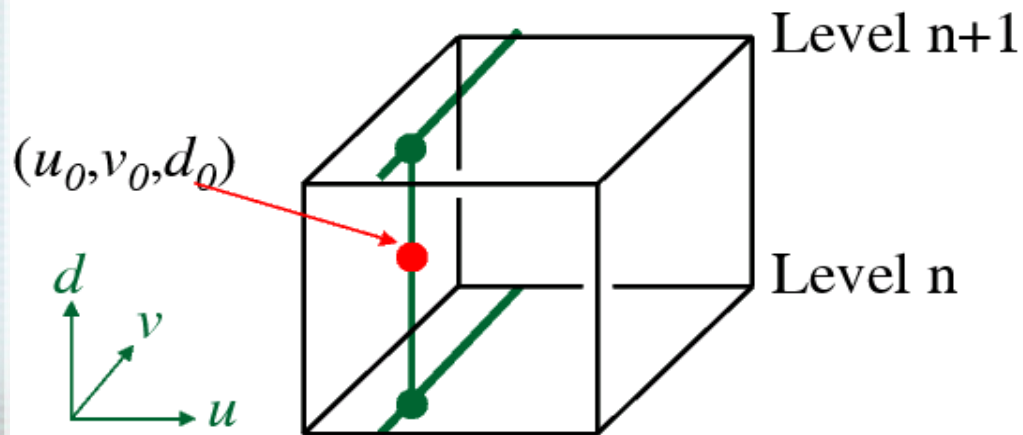
FilterMode:

```
LINEAR  
LINEAR_MIPMAP_LINEAR  
LINEAR_MIPMAP_NEAREST  
NEAREST  
NEAREST_MIPMAP_NEAREST
```

# Texturing

## Mipmapping

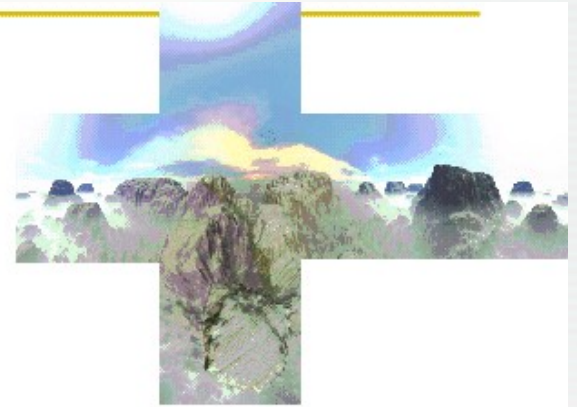
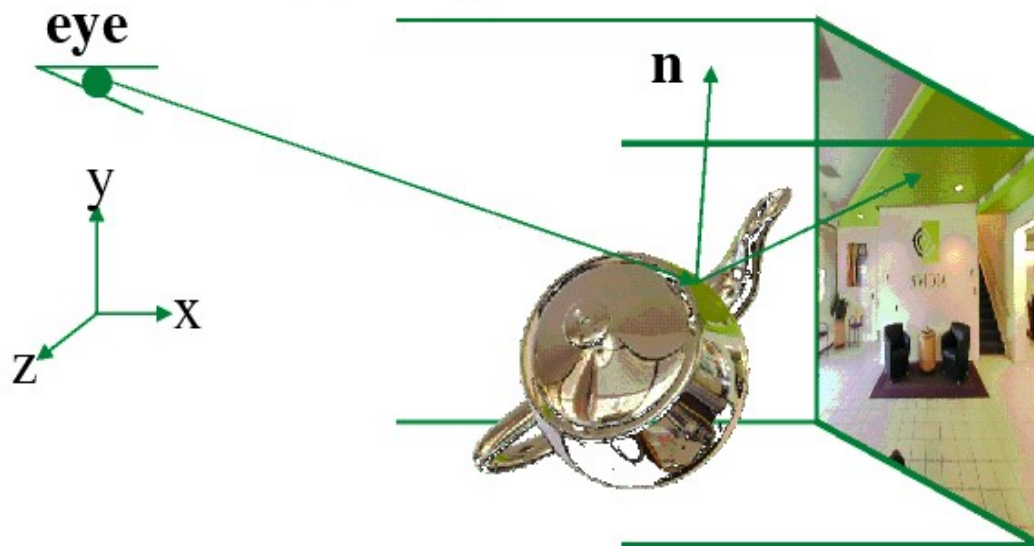
- Interpolate between those bilinear values
  - Gives trilinear interpolation



- Constant time filtering: 8 texel accesses
- How to compute  $d$ ?

# Texturing

## Environment mapping - Cube mapping



- Simple math: compute reflection vector,  $\mathbf{r}$
- Largest abs-value of component, determines which cube face.
  - Example:  $\mathbf{r}=(5,-1,2)$  gives POS\_X face
- Divide  $\mathbf{r}$  by 5 gives  $(u,v)=(-1/5,2/5)$
- If your hardware has this feature, then it does all the work

# Texturing

## Environment Mapping

```
osg::TexGen* texgen = new osg::TexGen;  
texgen->setMode(osg::TexGen::SPHERE_MAP);  
  
osg::StateSet* stateset = new osg::StateSet;  
stateSet->setTextureAttributeAndModes(0, texture, osg::StateAttribute::ON);  
  
stateSet->setTextureAttributeAndModes(0, texgen, osg::StateAttribute::ON);
```