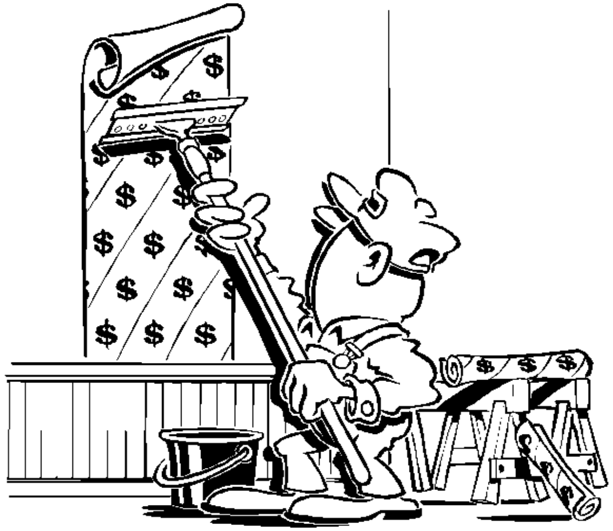
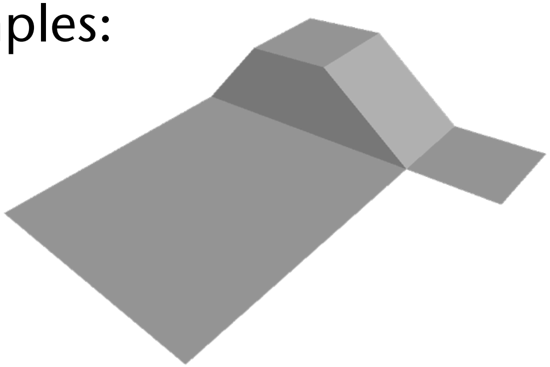
Advanced Computer Graphics
Advanced Texturing Methods

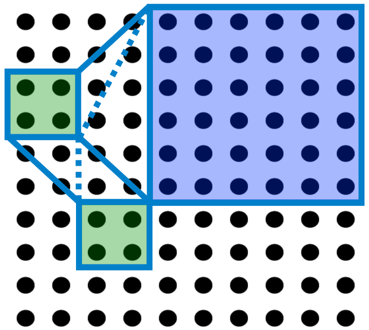


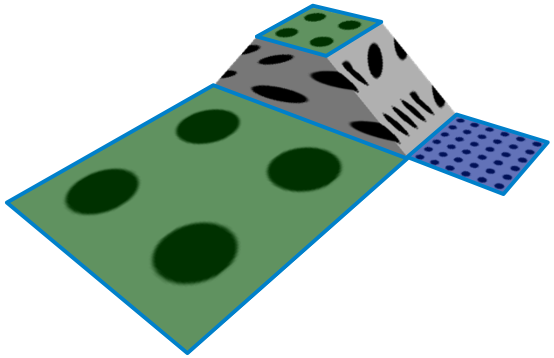
G. Zachmann
University of Bremen, Germany
cgvr.informatik.uni-bremen.de

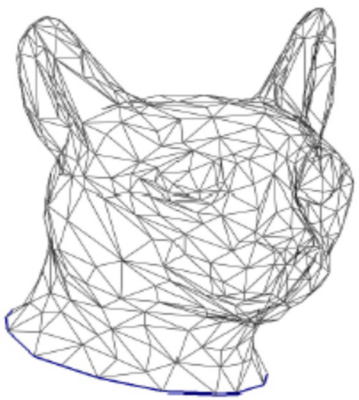
Problems with (Simple) Parameterizations

- Distortions in size & form
- Consequence: relative over- or under-sampling
- Examples:

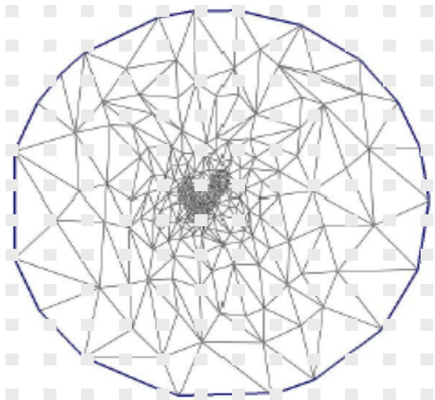








Mesh

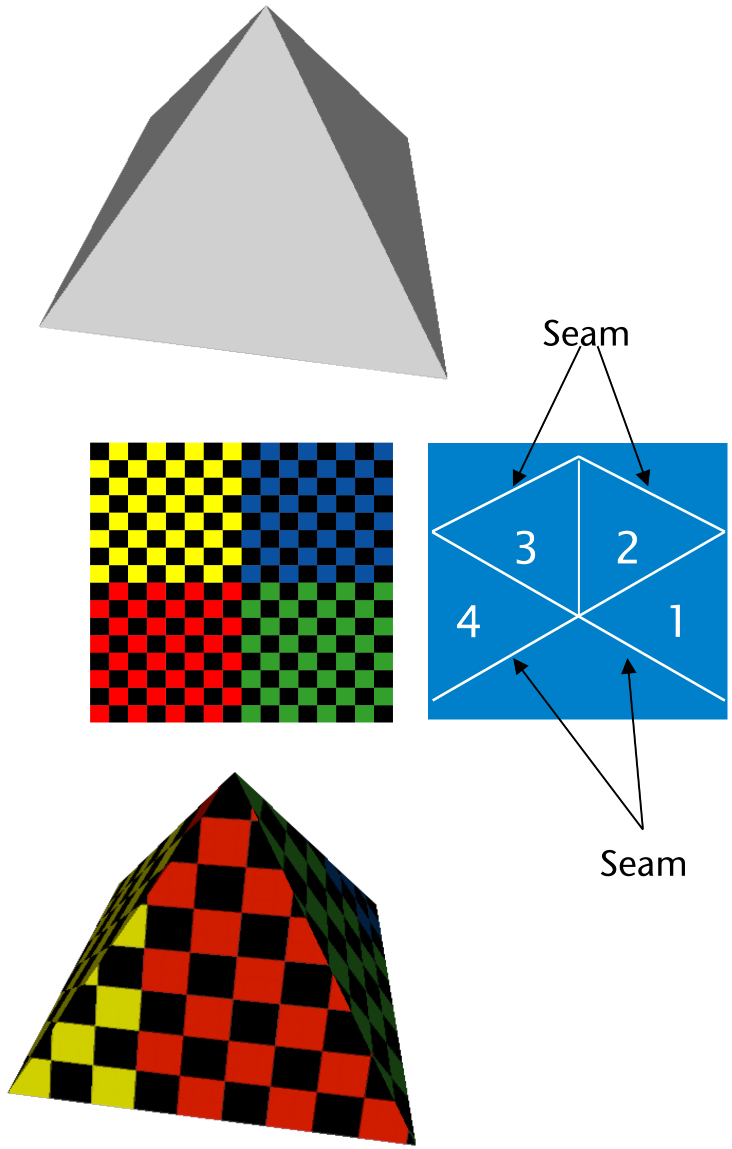


Embedding

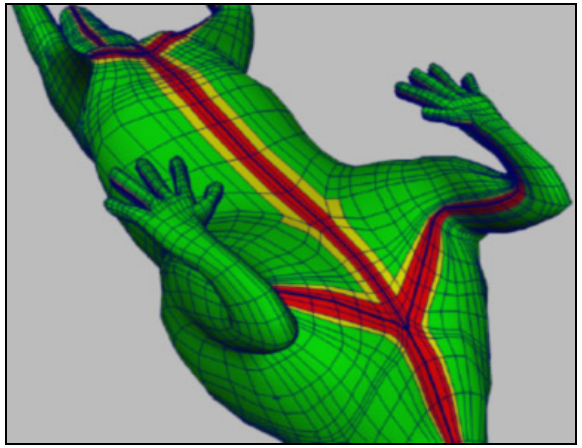
Distortion

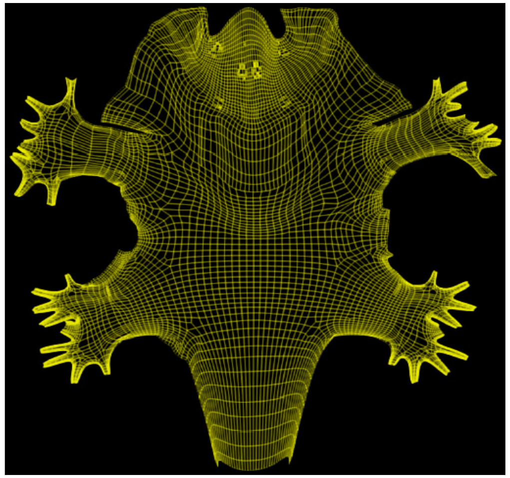
One Technique: Seams ("Nähte", Textursprünge)

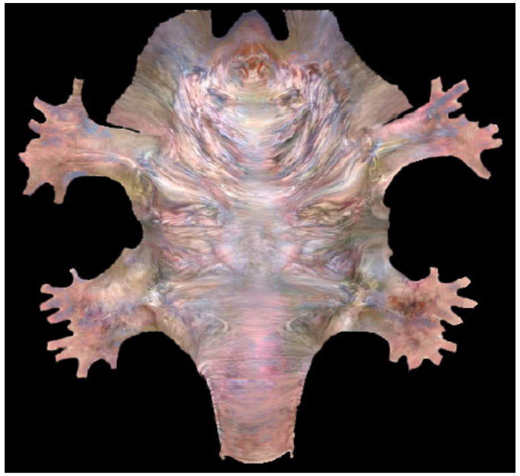
- Goal: minimize the distortion
- Idea: cutting up the mesh along certain edges
- Results in "double edges", also called *seams*
- Unavoidable with non-planar topology



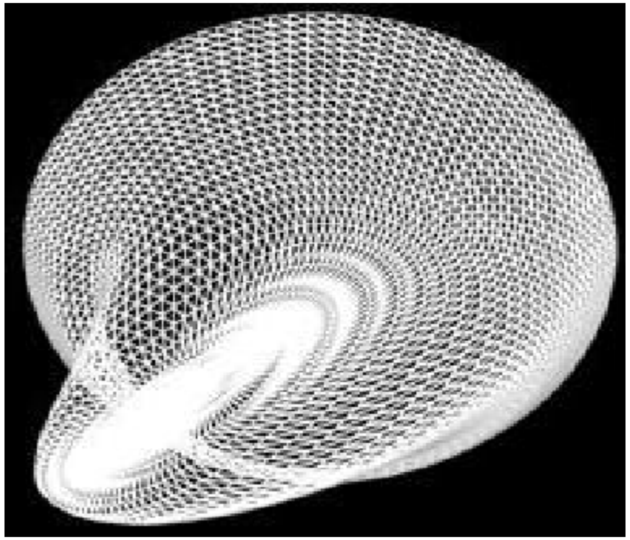
- Cut the object along only one continuous edge (preferably at inconspicuous places)
- Effect: the resulting mesh is now topologically equivalent to a disc
- Then embed this cut-open mesh into the 2D plane

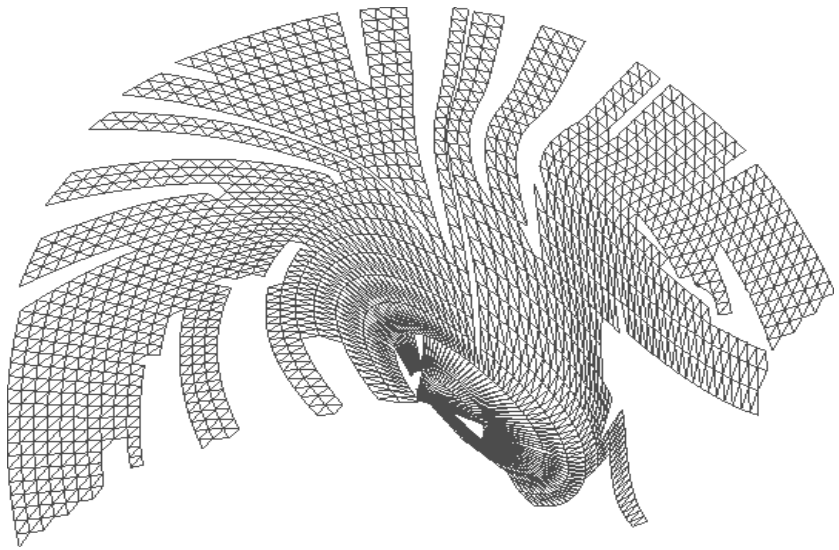






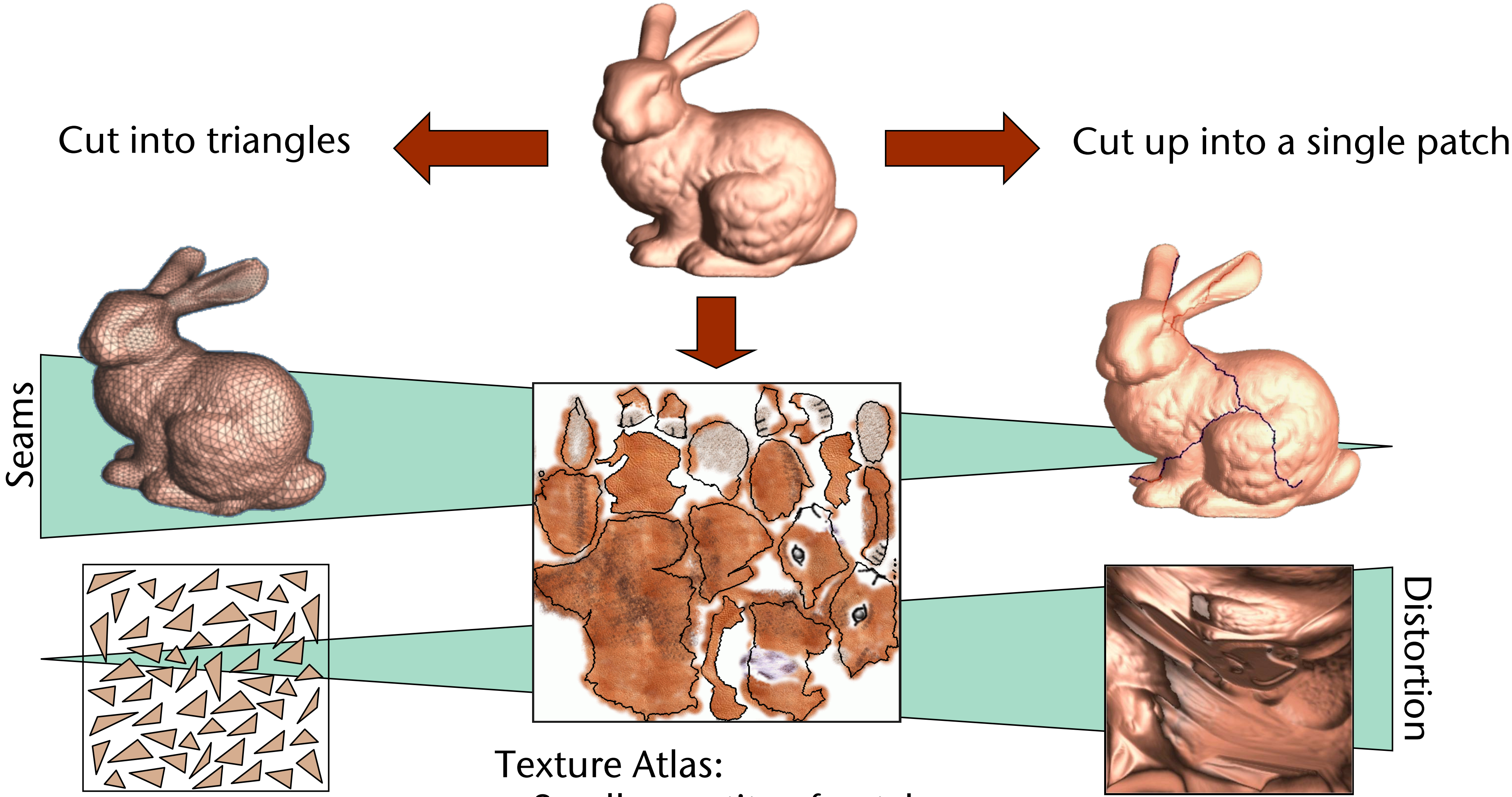
- Problem: there are still distortions
- Straight-forward remedy: multiple incisions
  - Problem: produces a severely fragmented embedded grid with many seams





- Another problem with seams: vertices on the seam must have multiple (u,v) coordinates
- Remedy: create multiple copies of those vertices
- New problem in case of deformations of the mesh

Distortion or Seams?

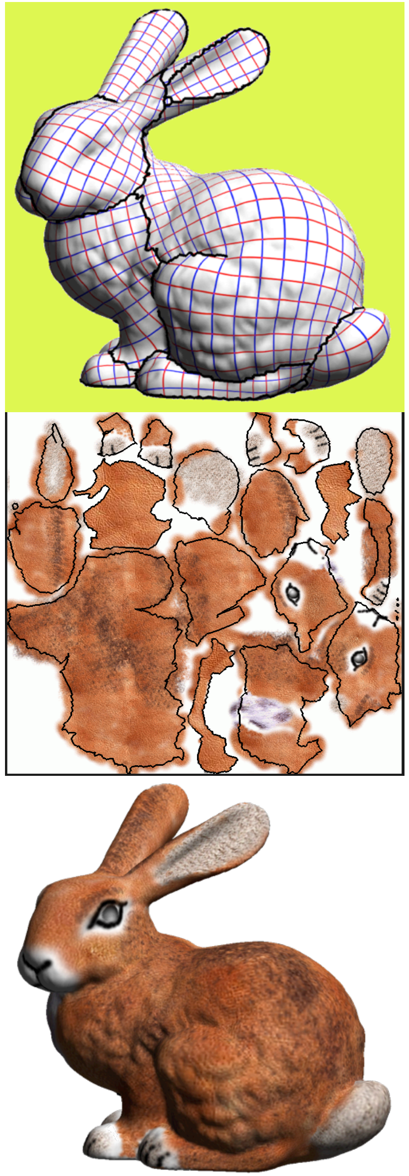


Texture Atlas:

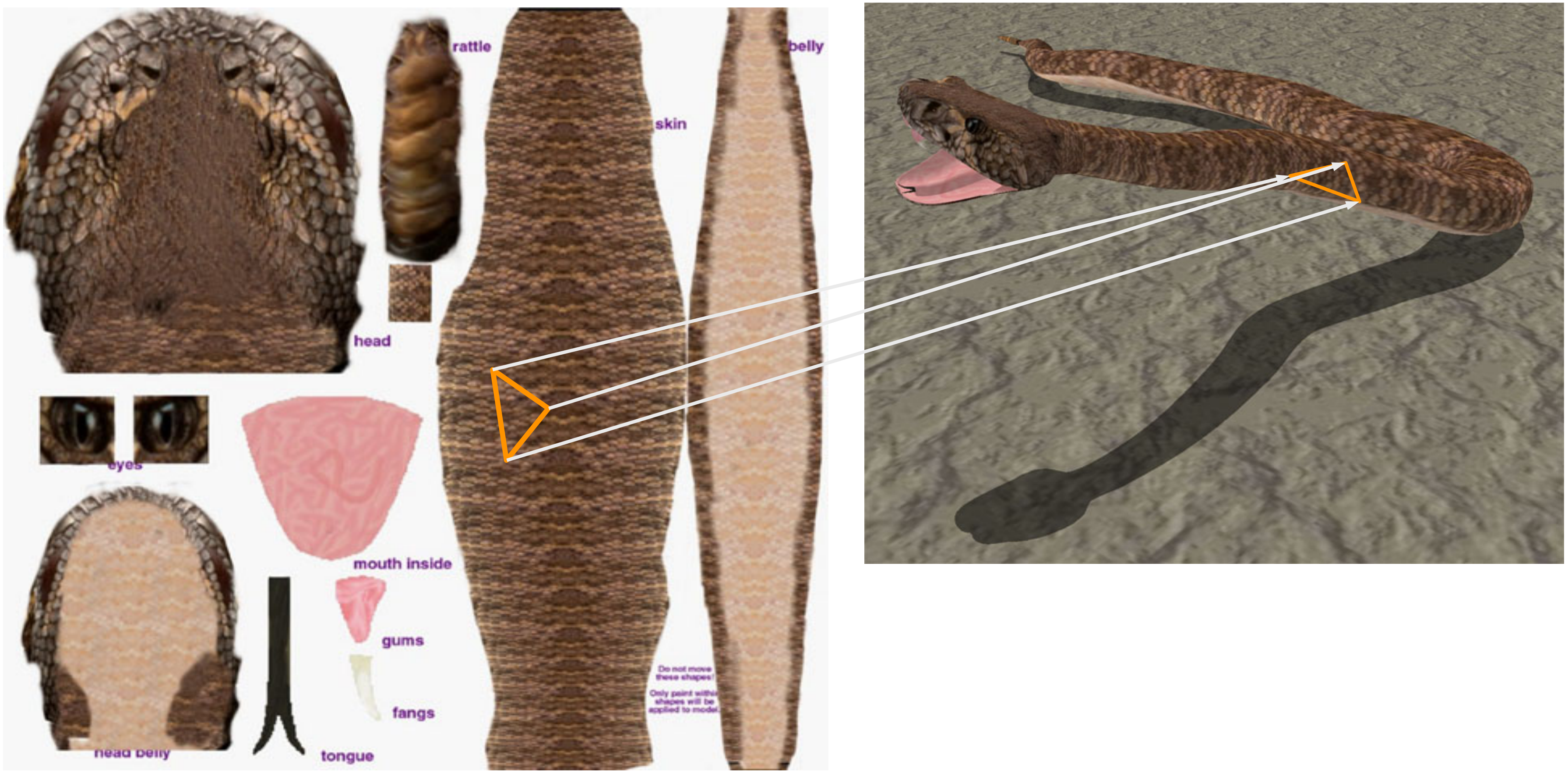
- Small quantity of patches
- Short and hidden seams

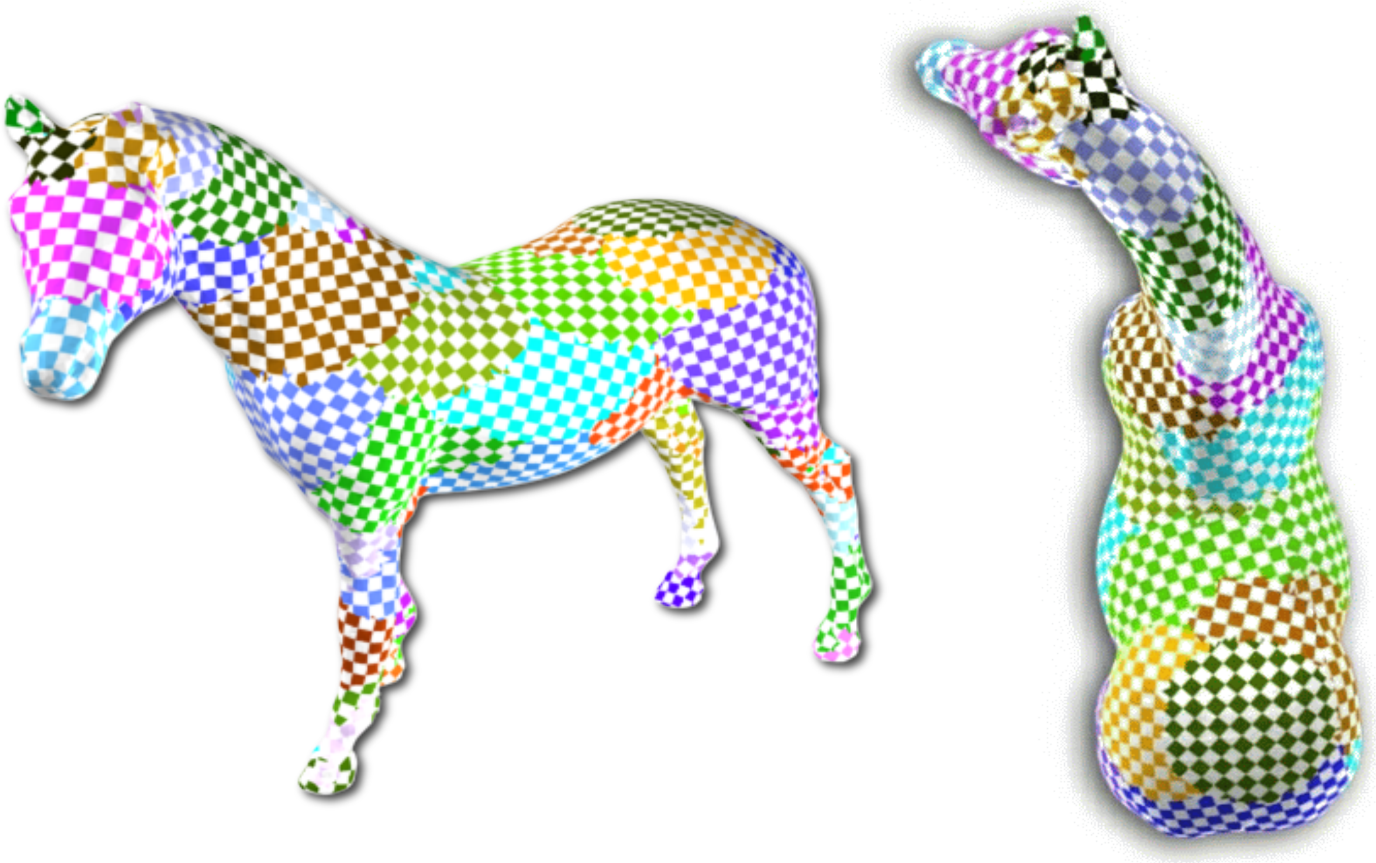
Texture Atlas

- Idea:
  - Cut the 3D surface in individual patches
  - Map = individual parameter domain in texture space for a single patch
  - Texture Atlas = set of these patches with their respective maps (= parameter domains)
- Statement of the problem:
  - Choose a compromise between seams and distortion
  - Hide the cuts in less visible areas
    - How do you do that automatically?
  - Determine a compact arrangement of texture patches (a so-called *packing problem*)



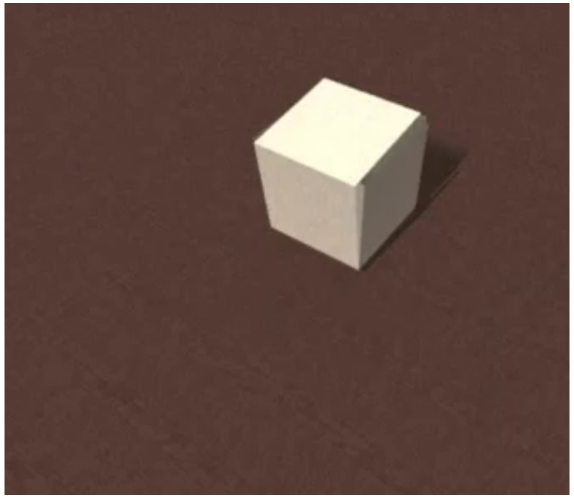
- Example:


rattle
belly
skin
head
eyes
mouth inside
gums
fangs
head belly
tongue

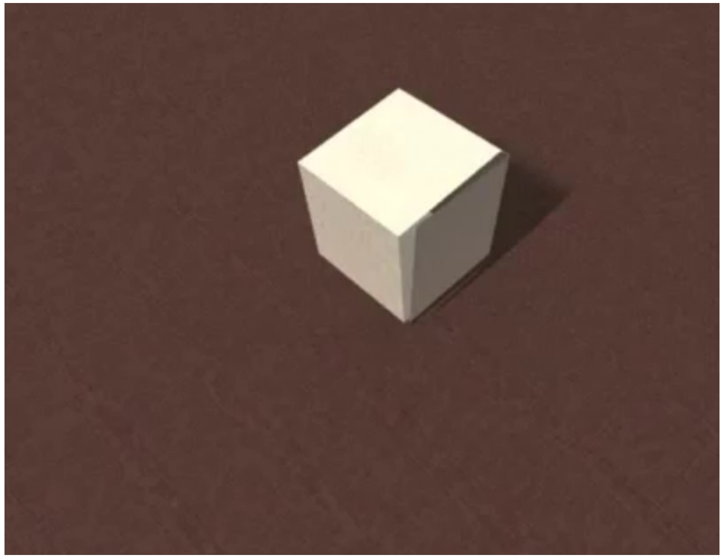


Digression: A Geometric Brain-Teaser

- A cube can be unfolded into a cross:



- Into what other forms can a cube be unfolded, too?



- Side note: the (unfolded) cube can be folded into a parallelogram

- BTW: all platonic solids except for the dodecahedron can be folded into a parallelogram in this way …

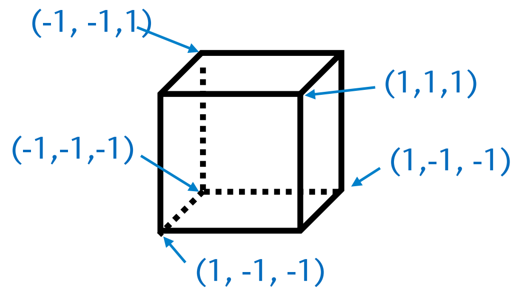
Cube Maps

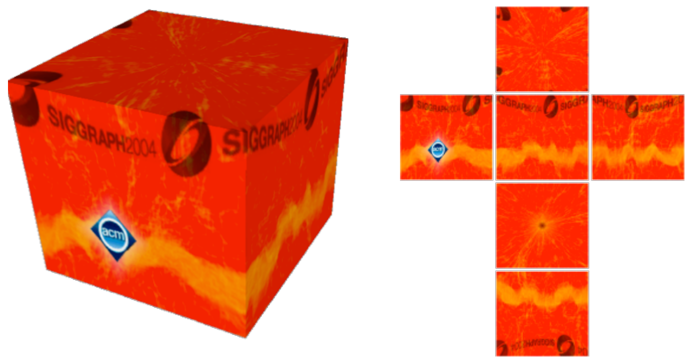
[Greene '86, Voorhies '94]

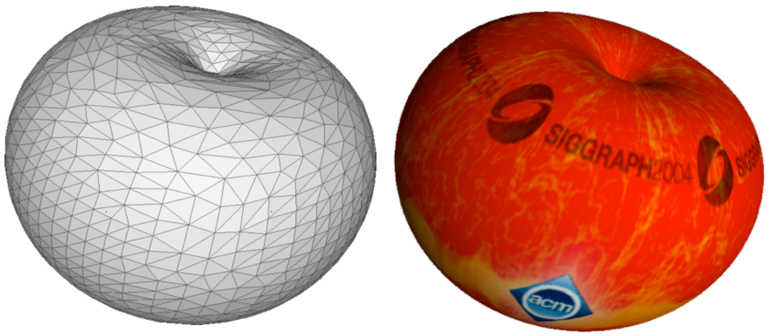
- Parameter domain Ω = unit cube:
  - Six quadratic texture bitmaps
  - 3D texture coordinates in OpenGL:

```
glTexCoord3f( s, t, r );
glVertex3f( x, y, z );
```

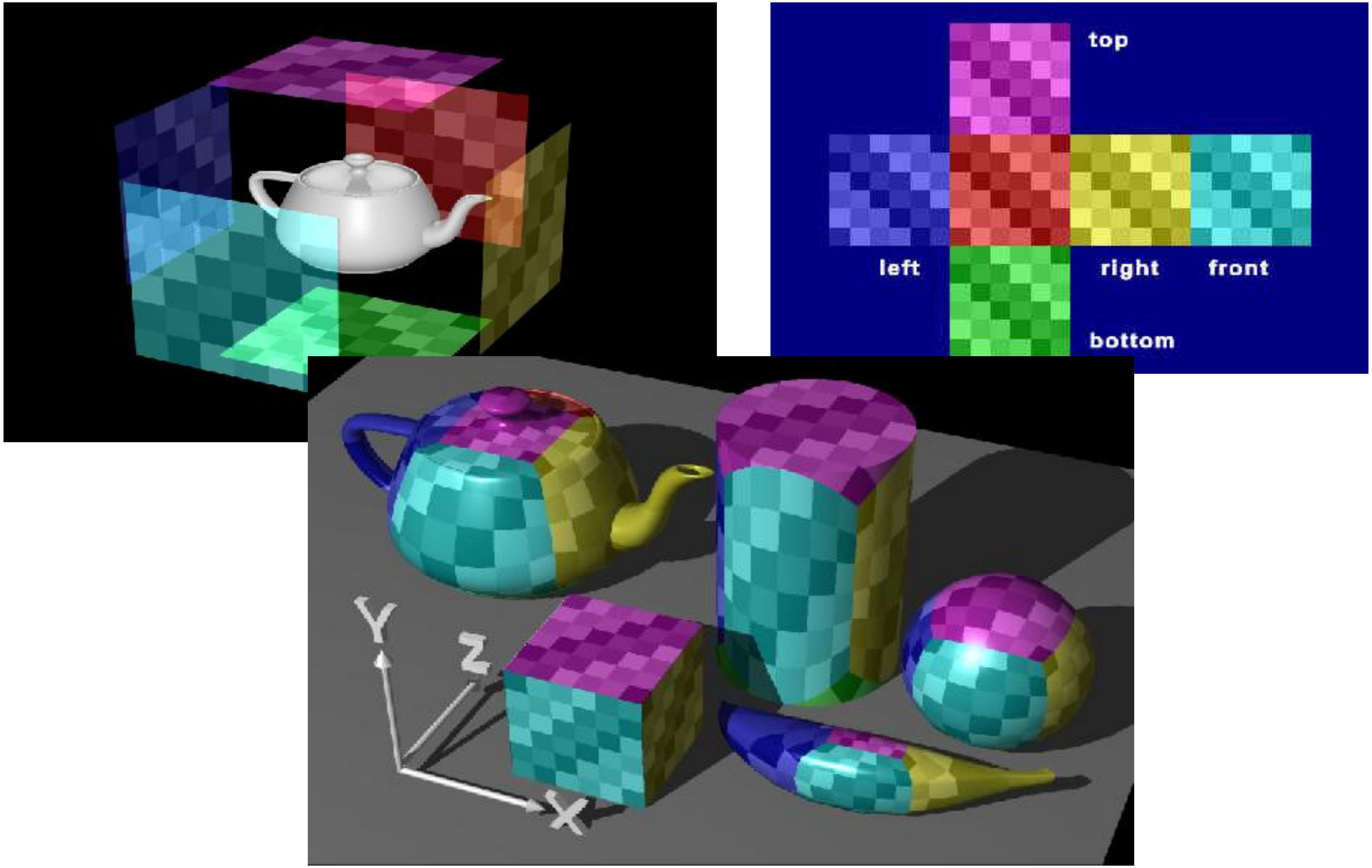
  - Largest component of (*s*,*t*,*r*) determines the map, intersection point determines (*u*,*v*) within the map
- Rasterization of cube maps:
  1. Interpolation of (*s*,*t*,*r*) in 3D
  2. Projection onto the cube ⟶ (*u*,*v*)
  3. Texture look-up in 2D
- Pro: relatively uniform, OpenGL support
- Slight con: one needs 6 images







Examples



Cube Maps in OpenGL

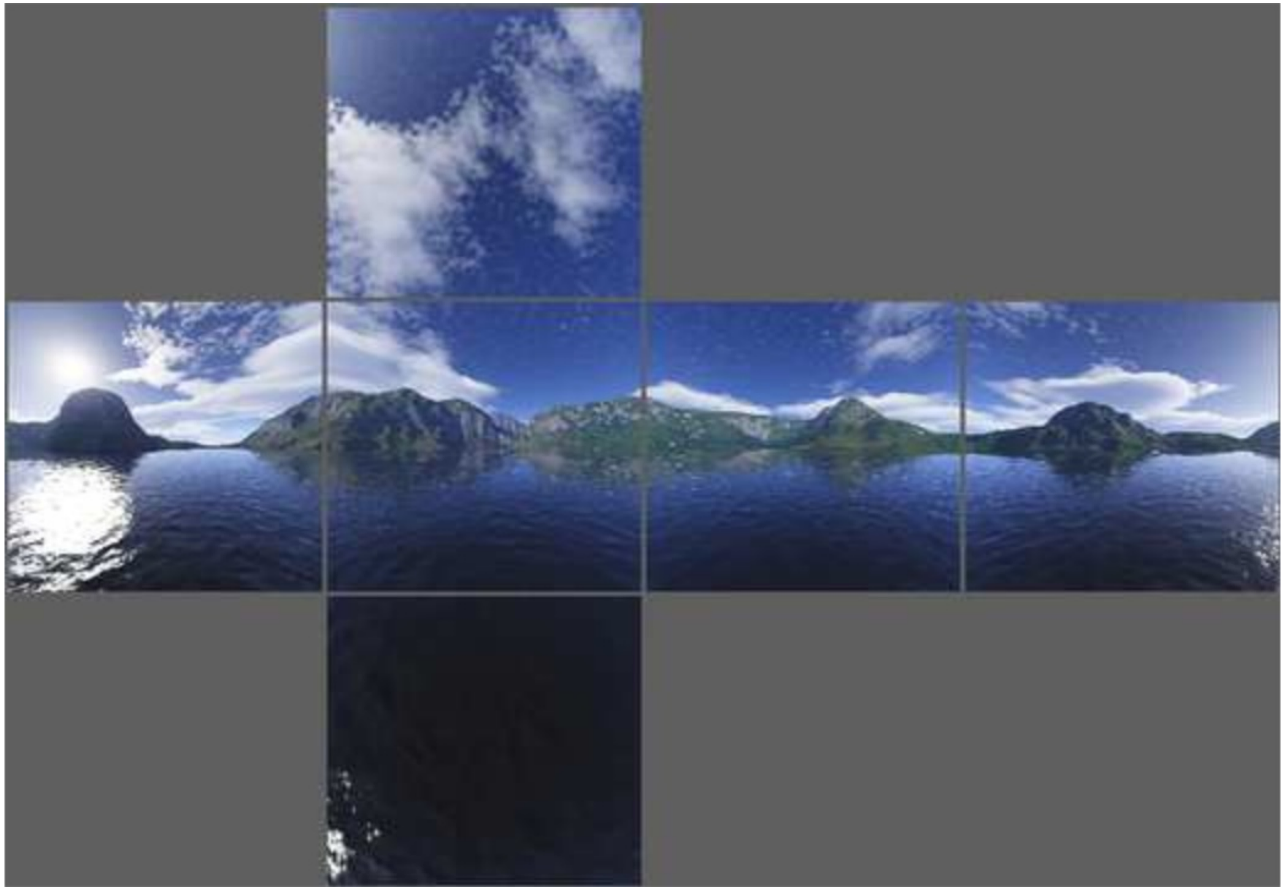
```
glGenTextures( 1, &textureID );
glBindTexture( GL_TEXTURE_CUBE_MAP, textureID );
glTexImage2D( GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGBA8, width, height,
              0, GL_RGB, GL_UNSIGNED_BYTE, pixels_face0 );
... Load the texture of the other cube faces
glTexParameteri( GL_TEXTURE_CUBE_MAP,
                 GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE );
... Set more texture parameters, like filtering

glEnable( GL_TEXTURE_CUBE_MAP );
glBindTexture( GL_TEXTURE_CUBE_MAP, textureID );
glBegin( GL_... );
glTexCoord3f( s, t, r );
glVertex3f( ... );
...
```

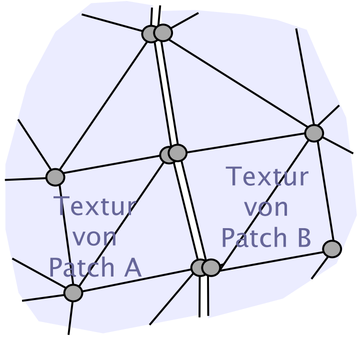
Analog:
`GL_TEXTURE_MAG_FILTER`,
`GL_TEXTURE_WRAP_T`, etc. …

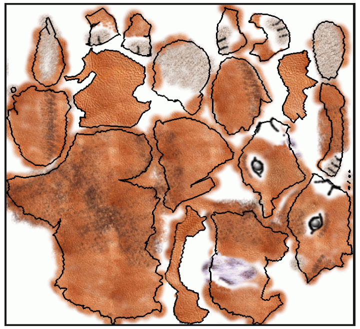
Just like with all other vertex attributes in OpenGL: first send all attributes, then the coordinates

- Example cube map for a sky box:

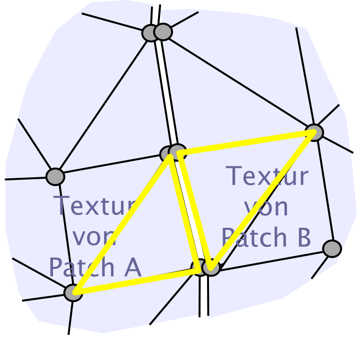


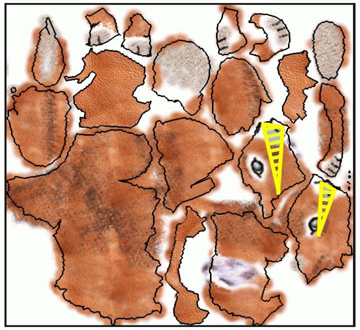
Texture Atlas vs. Cube Map





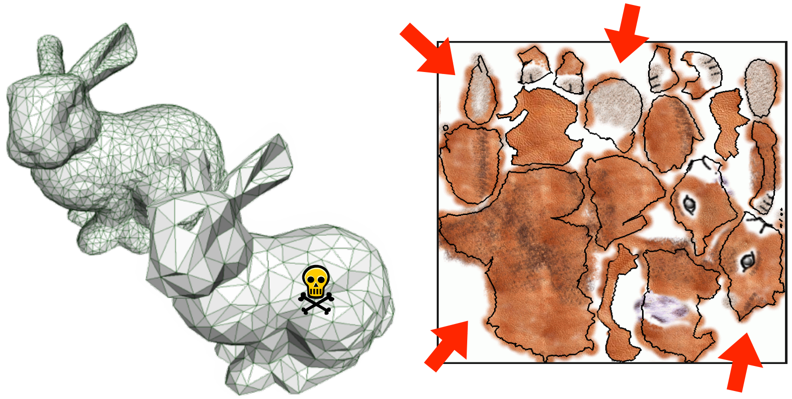
Texture Atlas vs. Cube Map

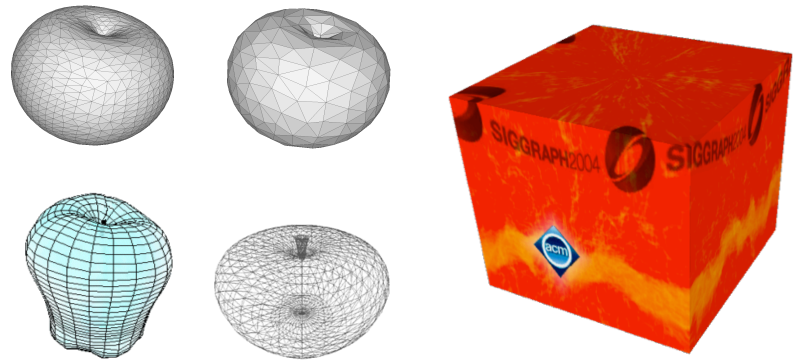




- Must prevent seams manually
  - E.g., by making colors match across seams
- MIP-mapping is difficult

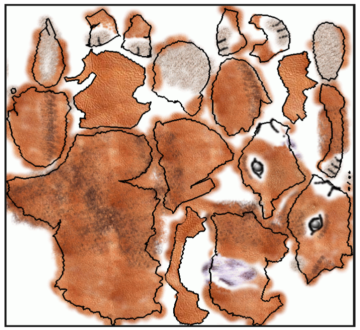
- No seams automatically
  - There are no gaps in the parameter domain
- MIP-mapping is okay

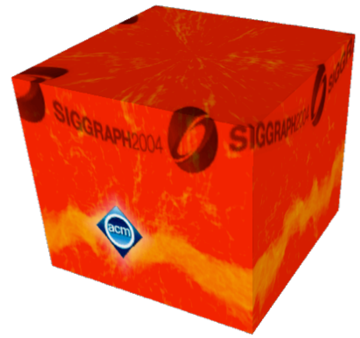




- Must prevent seams manually
- Triangles may lie within the patches
- MIP-mapping is difficult
- Only valid for a specific mesh
- Texels are wasted

- No seams automatically
- Triangles can lie in multiple patches
- MIP-mapping is okay
- Valid for many meshes
- All texels are used





- Must prevent seams manually
- Triangles may lie … the patches
- MIP-map … difficult
- Only … r a specific mesh
- T… e wasted

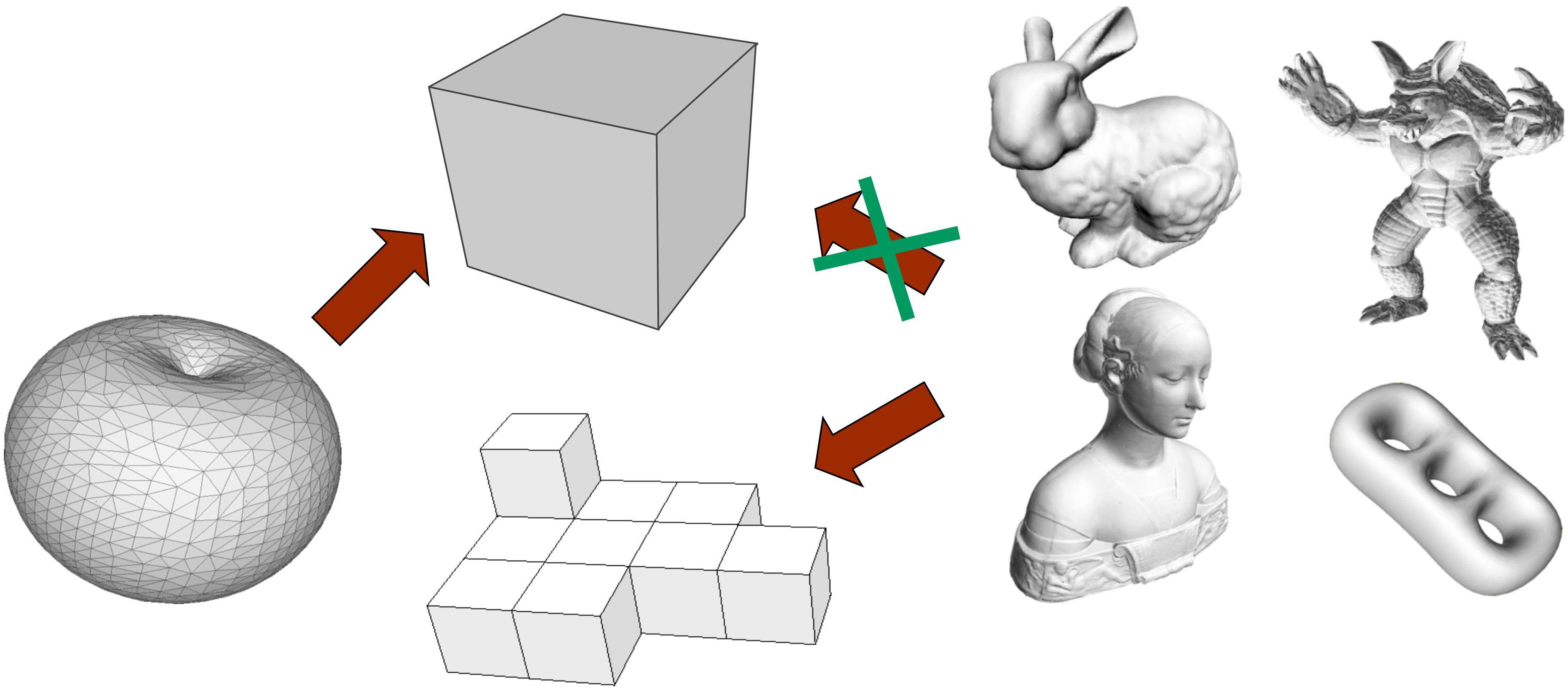
Works for any mesh

- No seams automati…
- Triangles can lie … patches
- MIP-ma…
- Vali… shes
- A… used

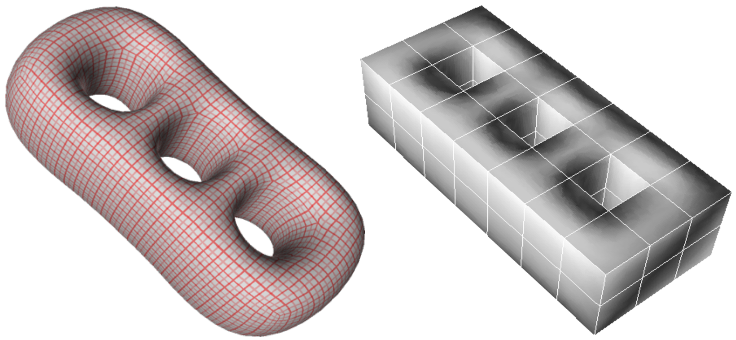
Only for "spheres-like" objects

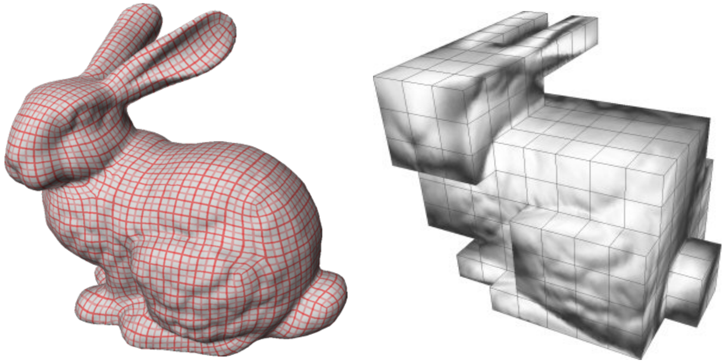
Polycube Maps

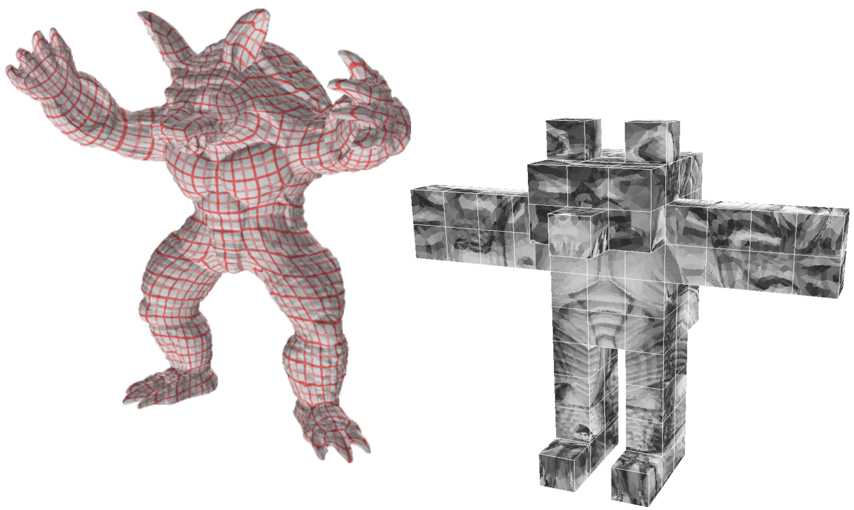
- Use many cube maps instead of an individual cube ⟶ polycube map
- Adapted to geometry and topology

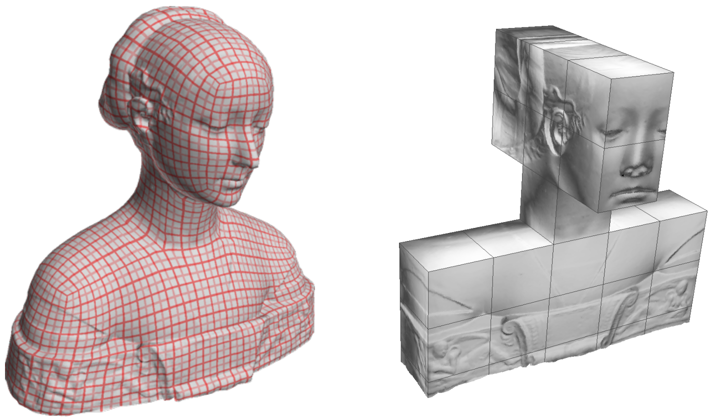


Examples



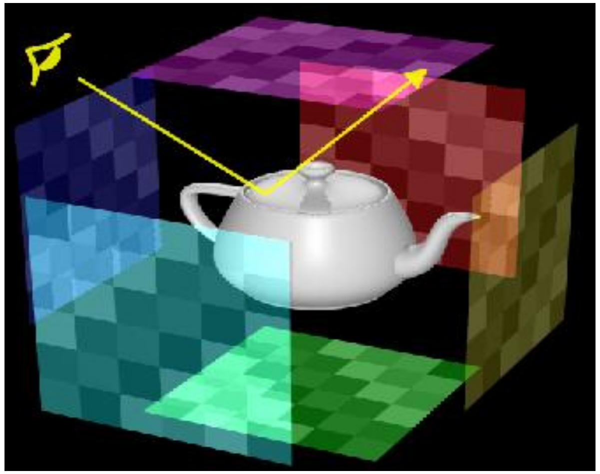




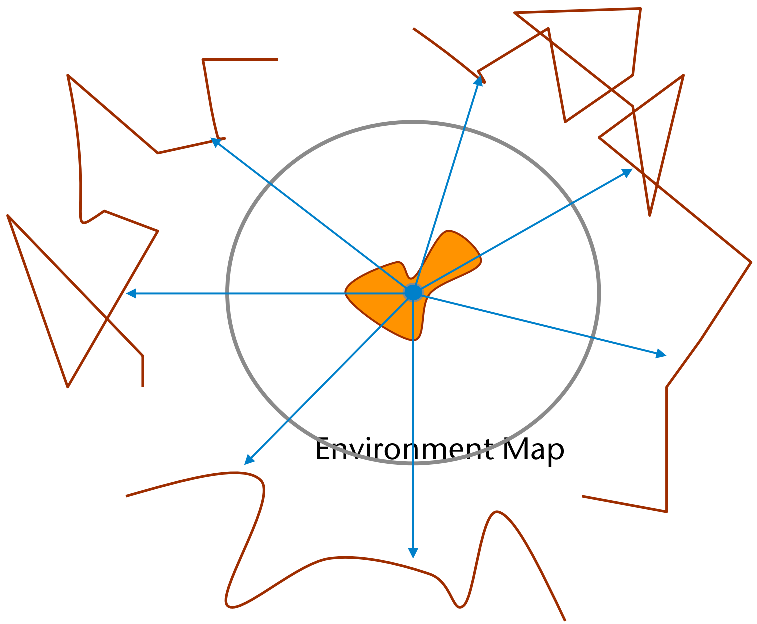


Environment Mapping

- With very reflective objects, one would like to see the surrounding environment reflected in the object
- Ray-tracing can do this, but not the polygonal rendering by rasterization
- The idea of environment mapping:
  - "Photograph" the environment in a texture
  - Save this in a so-called environment map
  - Use the reflection vector (from the ray) as an index in the texture
  - A.k.a. reflection mapping



- For every spatial direction, the environment map saves the color of the light that reaches a specific point
- Only correct for *one* position
- No longer correct if the environment changes



Historical Examples of Applications

Lance Williams, Siggraph 1985

Flight of the Navigator in 1986;
first feature film to use the technique

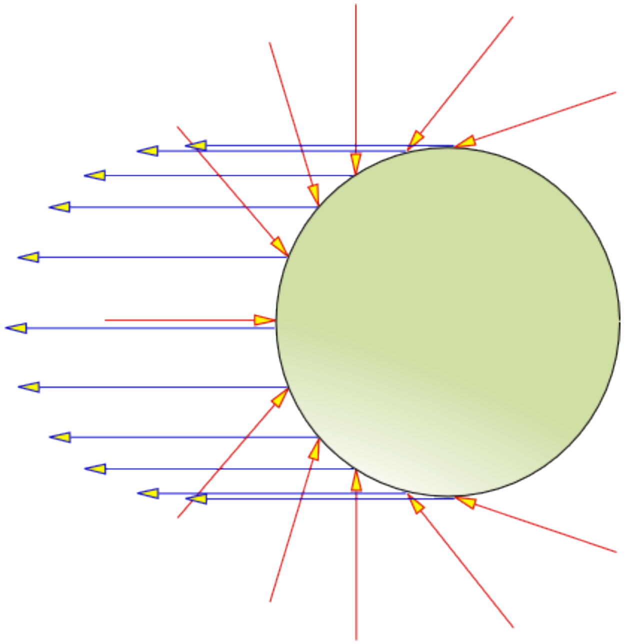
Terminator 2: Judgment Day - 1991
most visible appearance — Industrial Light + Magic

Environment Mapping Steps

- Generate or load a 2D texture that depicts the environment
- During rasterization, for every pixel of the reflected object...
  1. Calculate the normal **n**
  2. Calculate a reflection vector **r** from **n** and the view vector **v**
  3. Calculate texture coordinates (u,v) from **r**
  4. Color the pixel with the texture value
- The problem: how does one parameterize the space of the reflection vectors?
  - I.e.: how does one map spatial directions (= 3D vectors) onto [0,1]x[0,1]?
- Desired characteristics:
  - Uniform sampling (number of texels per solid angle should be "as constant as possible" in all directions)
  - View-independent ⟶ only one texture for all camera positions
  - Hardware support (texture coordinates should be easy to generate)

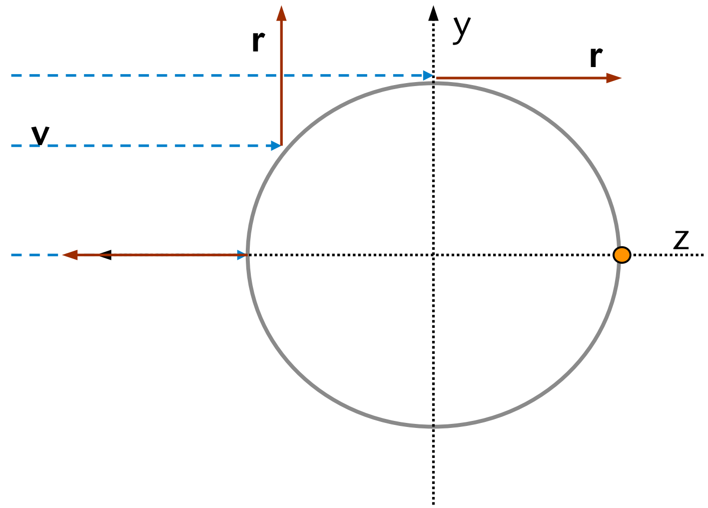
Spherical Environment Mapping

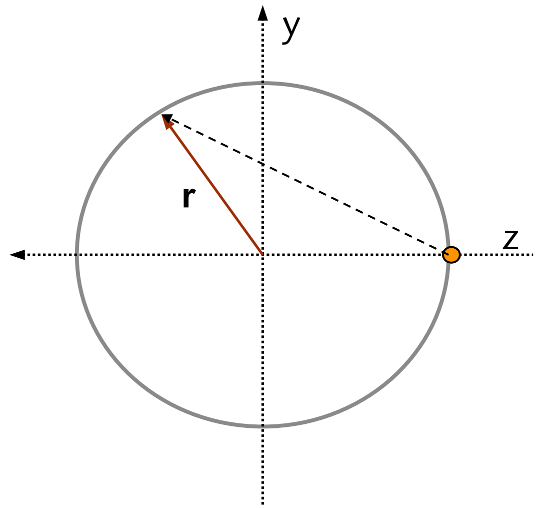
- Generating the environment map (= texture):
  - Photography of a reflective sphere; or
  - Ray-tracing of the scene with all primary rays being reflected at a perfectly reflective sphere



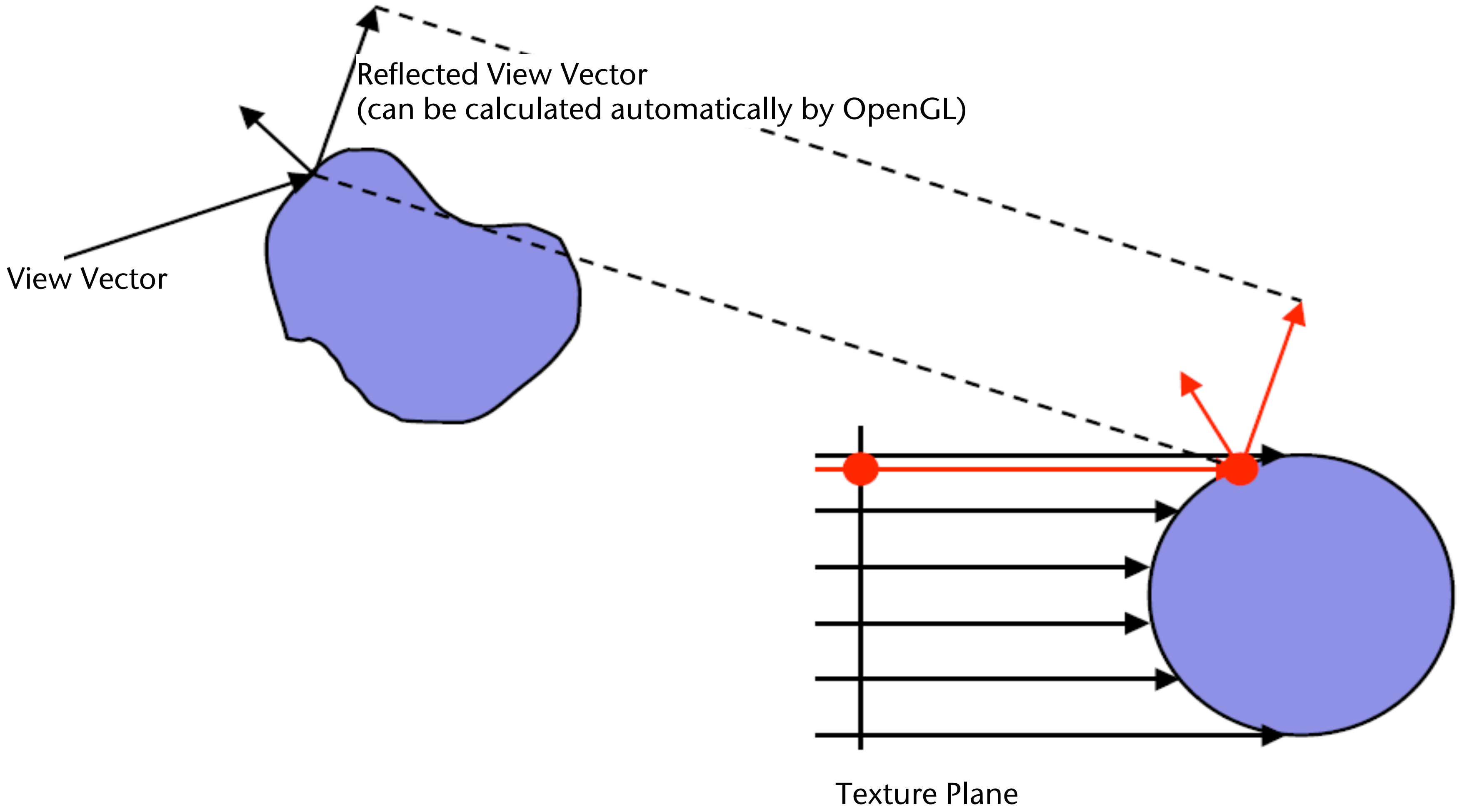
- Mapping of the directional vector **r** onto (u,v):
  - The sphere map contains (theoretically) a color value for every direction, except **r** = (0, 0, -1)
  - Mapping:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \frac{r_x}{\sqrt{r_x^2+r_y^2+(r_z+1)^2}} + 1 \\ \frac{r_y}{\sqrt{r_x^2+r_y^2+(r_z+1)^2}} + 1 \end{pmatrix}$$



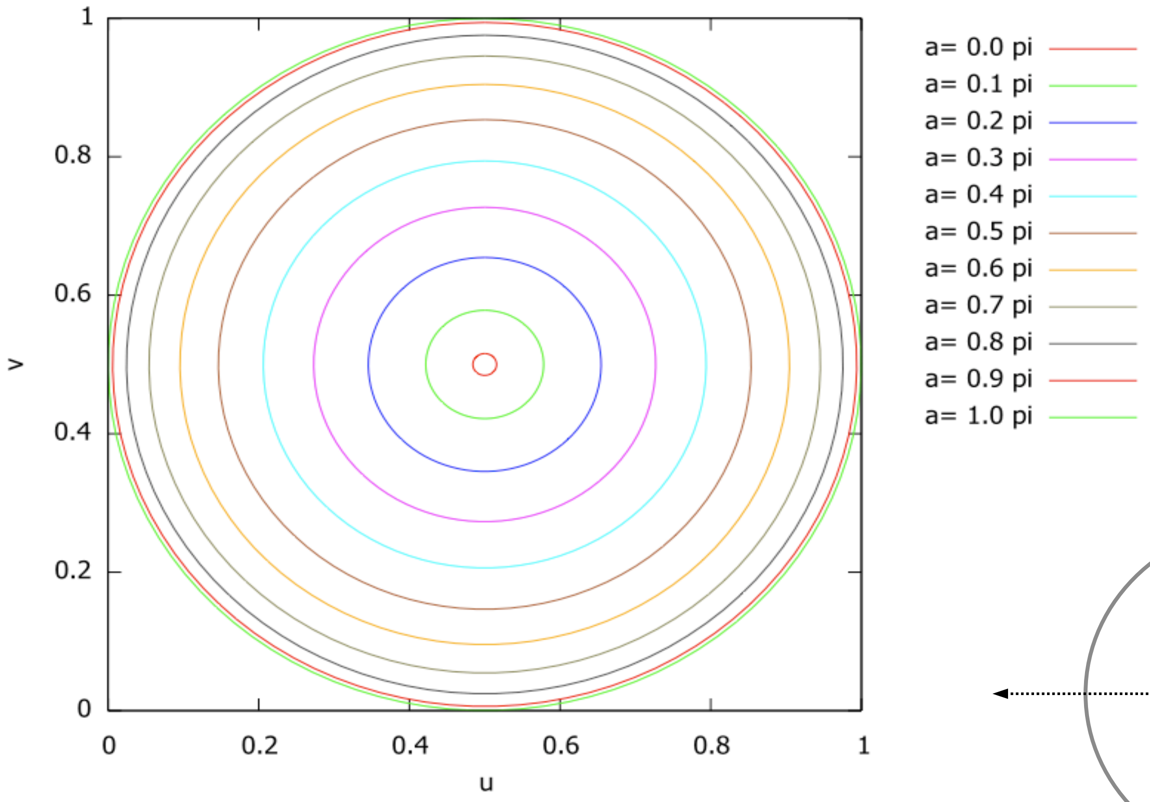


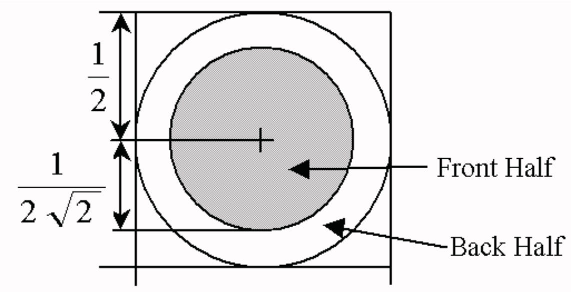
- Application of the sphere mapping to texturing:

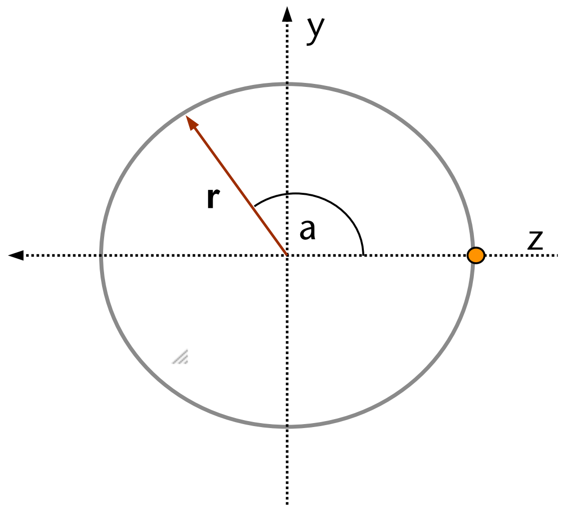


Simple Example

- Unfortunately, the mapping/sampling is not very uniform:

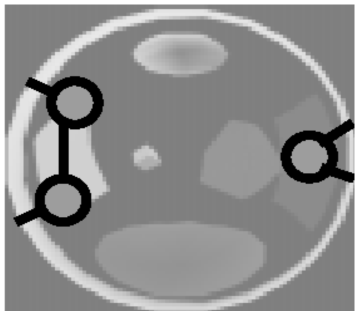


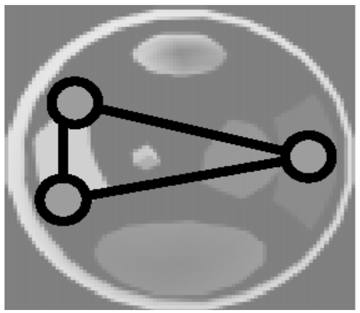




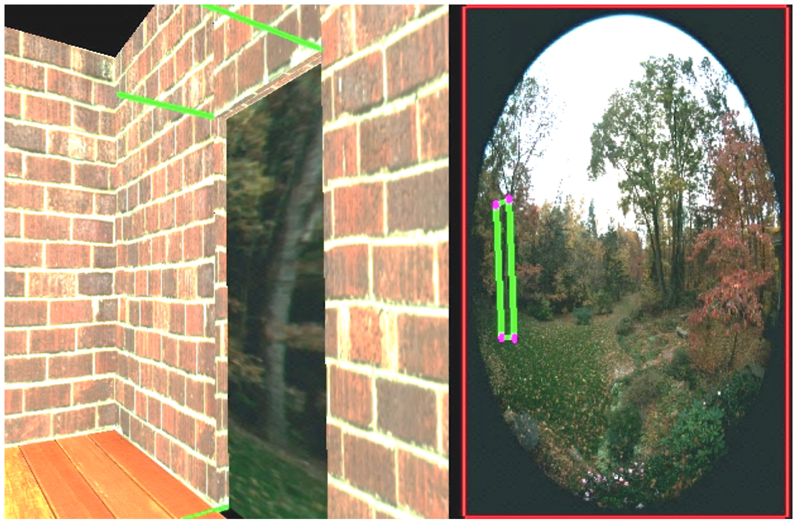
- Texture coords are interpolated incorrectly:
  - Texture coords are interpolated linearly (by the rasterizer), but the sphere map is non-linear
  - Long polygons can cause serious "bends" in the texture
  - Sometimes, incorrect wrap-arounds occur with interpolated texture coords
  - *Sparkles / speckles* if the reflecting vector comes close to the edge of the texture (through aliasing and "wrap-around")

Intended/ correct wrap through the sphere perimenter





2D texturing hardware doesn't know about sphere maps, it just linearly interpolates texture coords



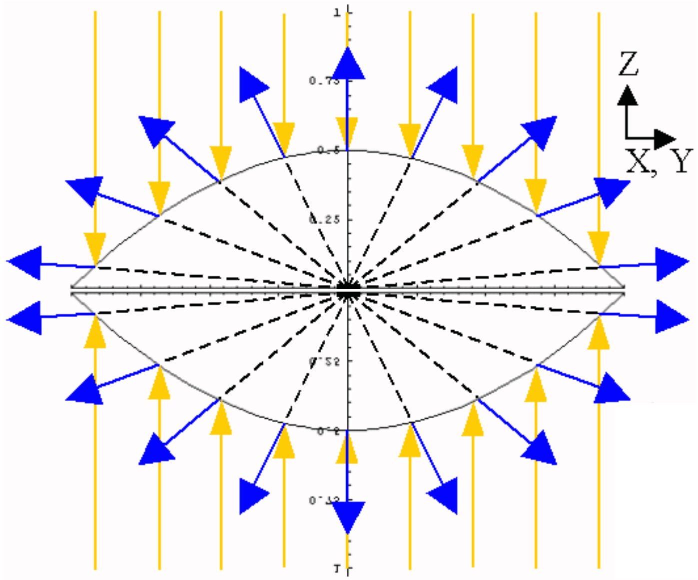
- Other cons:
  - Textures are difficult to generate by program (other than ray-tracing)
  - *Viewpoint dependent*: the center of the spherical texture map represents the vector that goes directly back to the viewer!
    - Can be made *view independent* with some OpenGL extensions
- Pros:
  - Easy to generate texture coordinates
  - Supported in OpenGL

A Piece of Artwork

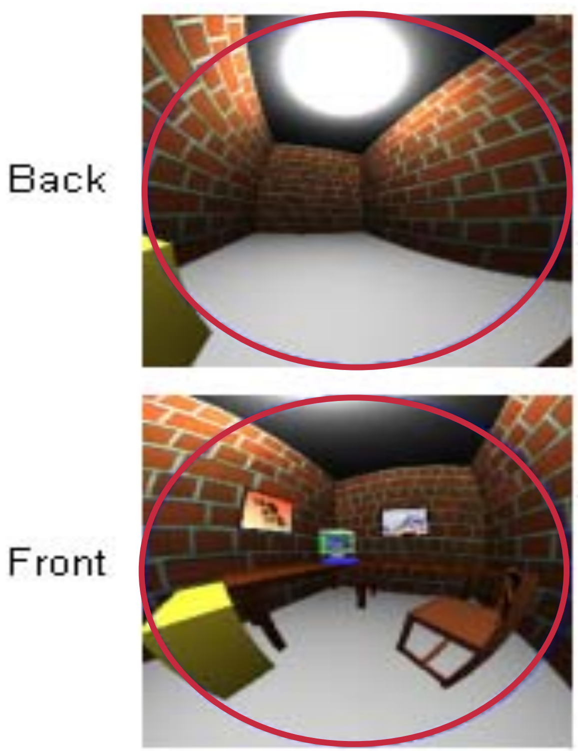
Reflective balls in the main street of Adelaide, Australia

Dual Parabolic Environment Mapping

- Idea:
  - Map the environment onto two textures via a reflective double paraboloid
  - Pros:
    - Relatively uniform sampling
    - *View independent*
    - Relatively simple computation of texture coordinates
    - Also works in OpenGL
    - Also works in a single rendering pass (just needs multi-texturing)
  - Cons:
    - Produces artifacts when interpolating across the edge



- Images of the environment (= directional vectors) are still discs (as with the sphere map)
- Comparison:

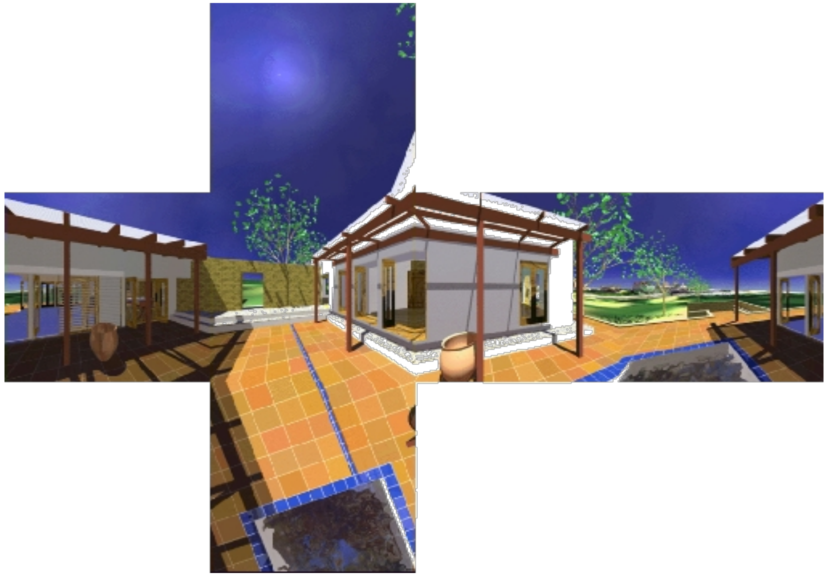


Parabolic environment map

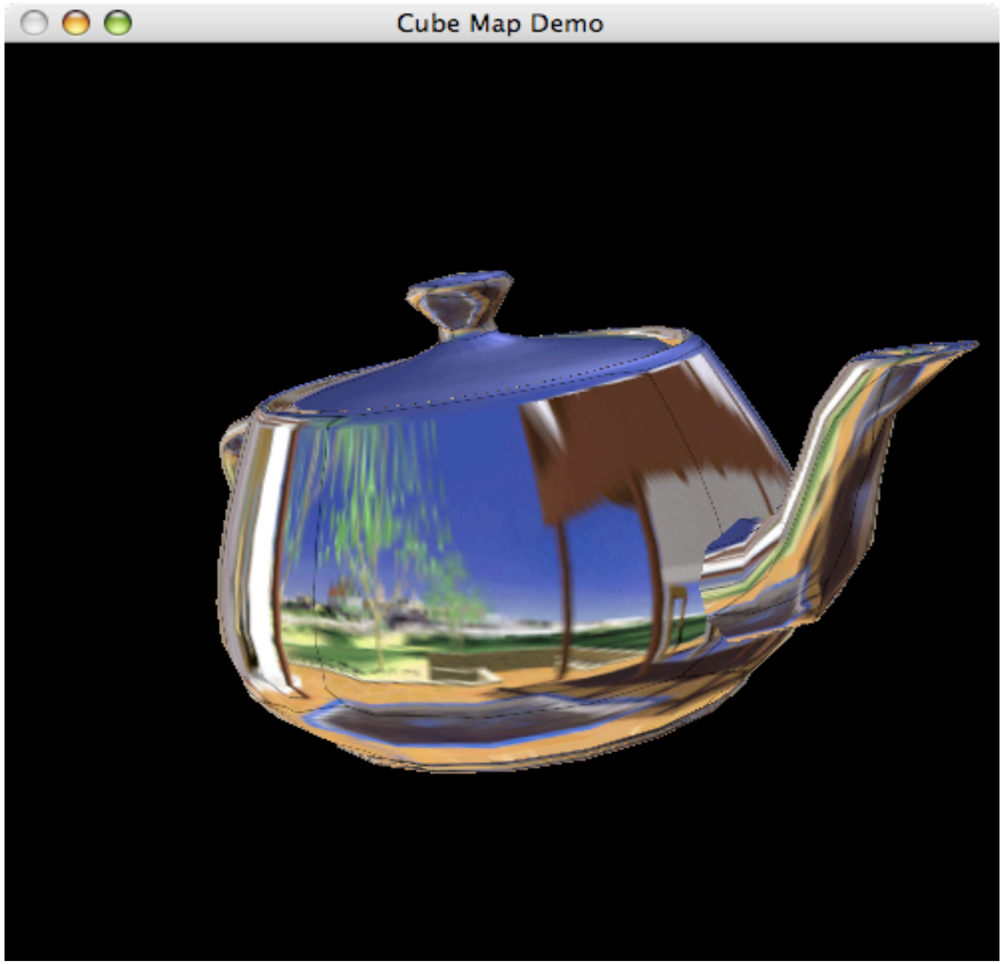
Result

Cubic Environment Mapping

- As before with the "normal" cube maps
- Only difference: use the reflected vector **r** for the calculation of the texture coordinates
- This reflected vector can be automatically calculated by OpenGL for each vertex (**`GL_REFLECTION_MAP`**)



Demo with Static Environment

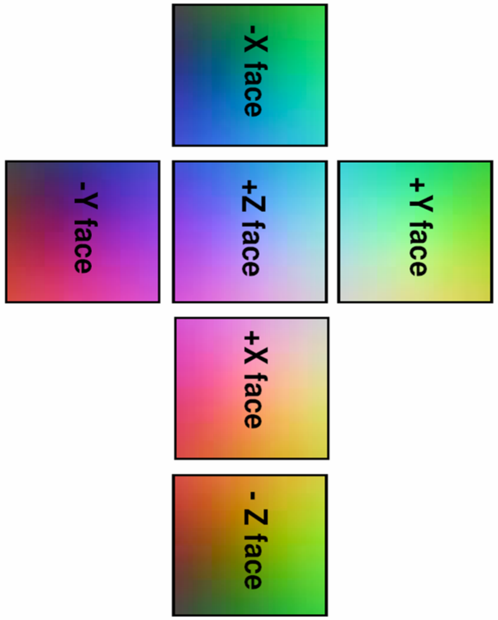


Cube Maps as LUT for Directional Functions

- Further application: one can also use a cube map to store any function of direction! (as a precomputed lookup table)
- Example: normalization of a vector
  - Every cube map texel (*s*,*t*,*r*) stores this vector

$$\frac{(s,t,r)}{\|(s,t,r)\|}$$

    in its RGB channels
  - Now one can specify any texture coordinates using **`glTexCoord3f()`** and receives the normalized vector
- Warning: when using this technique, one should turn off filtering



Dynamic Environment Maps

- Until now: environment map was invalid as soon as something in the environmental scene had changed!
- Idea:
  - Render the scene from the "midpoint" outward (typically 6x for cube map)
  - Transfer framebuffer to texture (using the appropriate mapping)
  - Render the scene again from the viewpoint outward, this time with environment mapping
- ➢ Multi-pass rendering
- Typically used with cube env maps

Dynamic Environment Mapping in OpenGL Using Cube Maps

```
GLuint cm_size = 512;    // texture resolution of each face
GLfloat cm_dir[6][3];    // direction vectors
float dir[6][3] = {
    1.0, 0.0, 0.0,       // right
   -1.0, 0.0, 0.0,       // left
    0.0, 0.0, -1.0,      // bottom
    0.0, 0.0, 1.0,       // top
    0.0, 1.0, 0.0,       // back
    0.0, -1.0, 0.0       // front
};
GLfloat cm_up[6][3] =    // up vectors
{   0.0, -1.0,  0.0,     // +x
    0.0, -1.0,  0.0,     // -x
    0.0, -1.0,  0.0,     // +y
    0.0, -1.0,  0.0,     // -y
    0.0,  0.0,  1.0,     // +z
    0.0,  0.0, -1.0      // -z
};
GLfloat cm_center[3];    // viewpoint / center of gravity
GLenum cm_face[6] = {
    GL_TEXTURE_CUBE_MAP_POSITIVE_X,
    GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
    GL_TEXTURE_CUBE_MAP_NEGATIVE_Z,
    GL_TEXTURE_CUBE_MAP_POSITIVE_Z,
    GL_TEXTURE_CUBE_MAP_POSITIVE_Y,
    GL_TEXTURE_CUBE_MAP_NEGATIVE_Y
};
// define cube map's center cm_center[] = center of object
// (in which scene has to be reflected)
...
```

```
// set up cube map's view directions in correct order
for ( uint i = 0, i < 6; i + )
  for ( uint j = 0, j < 3; j + )
        cm_dir[i][j] = cm_center[j] + dir[i][j];

// render the 6 perspective views (first 6 render passes)
for ( unsigned int i = 0; i < 6; i ++ )
{
  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
  glViewport( 0, 0, cm_size,  cm_size );
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  gluPerspective( 90.0, 1.0, 0.1, ... );
  glMatrixMode( GL_MODELVIEW );
  glLoadIdentity();
  gluLookAt( cm_center[0], cm_center[1], cm_center[2],
             cm_dir[i][0], cm_dir[i][1], cm_dir[i][2],
             cm_up[i][0],  cm_up[i][1],  cm_up[i][2] );
  // render scene to be reflected
  ...
  // read-back into corresponding texture map
  glCopyTexImage2D( cm_face[i], 0, GL_RGB, 0, 0, cm_size, cm_size, 0 );
}
```

```
// cube map texture parameters init
glTexEnvf(  GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );
glTexParameteri( GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP );
glTexParameteri( GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP );
glTexParameterf( GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameterf( GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexGeni( GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP );
glTexGeni( GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP );
glTexGeni( GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP );

// enable texture mapping and automatic texture coordinate generation
glEnable( GL_TEXTURE_GEN_S );
glEnable( GL_TEXTURE_GEN_T );
glEnable( GL_TEXTURE_GEN_R );
glEnable( GL_TEXTURE_CUBE_MAP );

// render object in 7th pass ( in which scene has to be reflected )
...

// disable texture mapping and automatic texture coordinate generation
glDisable( GL_TEXTURE_CUBE_MAP );
glDisable( GL_TEXTURE_GEN_S );
glDisable( GL_TEXTURE_GEN_T );
glDisable( GL_TEXTURE_GEN_R );
```

Berechnet den Reflection Vector in Eye-Koord.

For Further Reading

- On the class's homepage:
  - "OpenGL Cube Map Texturing" (Nvidia, 1999)
    - With example code
    - Here several details are explained (e.g. the orientation)
  - "Lighting and Shading Techniques for Interactive Applications" (Tom McReynolds & David Blythe, Siggraph 1999);
  - SIGGRAPH '99 Course: "Advanced Graphics Programming Techniques Using OpenGL" (ist Teil des o.g. Dokumentes)