The Differential Geometry of Texture-Mapping and Shading

(work-in-progress)

Ken Turkowski
Interactive Media Lab
Advanced Technology Group
Apple Computer, Inc.

5 October 1993
(printed February 10, 1998)

1. The Differential Geometric Neighborhood of a Surface

For shading in three-dimensional computer graphics, it is useful to abstract the geometric properties of a surface at a point on that surface. The collection of properties we call the *surface neighborhood*, or simply the *neighborhood*.

1.1. Position

Given a point on a surface, its position or (x,y,z) coordinates can be found. It is usually already given.

1.2. Normal

The normal to a surface can be found for orientable surfaces. For parametric surfaces, that can be found by taking the cross product of the tangent in the u direction by the tangent in the v direction, and normalizing.

For implicit surfaces , $f(x,y,z) = 0$, the normal can be found by computing the gradient, $\mathbf{g} = \quad f$, and normalizing: $\mathbf{n} = \mathbf{g}/|\mathbf{g}|$.

Occasionally, there are singular points on a surface where the normal is multiply-defined (as at a crease or cusp or the apex of a cone), but we can always disambiguate them by association with a subsurface.

1.3. 2D Parametrization

Parametric surfaces are vector functions of two parameters: $(x,y,z) = \mathbf{p}(u,v)$. Sometimes auxiliary parametrizations are introduced, e.g. to assure that the parametrization lies in $[0,1] \times [0,1]$, which is a practical convention used for texture-mapping.

1.4. Tangents

Most surfaces we work with in computer graphics have some sort of parametrization, either natural or assigned. We can either evaluate the parametrization at each point or establish a consistent way to compute it. Either way, we can also evaluate the tangents to the surface in the direction of increasing parametrization. If the position is specified as $\mathbf{x}$=(x,y,z) and the parametrization is specified as (u,v), then the tangents are given by:

$$\frac{\partial(x,y,z)}{\partial u} = \begin{bmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \end{bmatrix} = \begin{bmatrix} x_u & y_u & z_u \end{bmatrix}$$

and

$$\frac{\partial(x,y,z)}{\partial v} = \begin{bmatrix} \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \end{bmatrix} = \begin{bmatrix} x_v & y_v & z_v \end{bmatrix}.$$

When these tangent vectors are stacked together to form a matrix, it is called the tangent Jacobian matrix:

$$\mathbf{T} = \frac{\partial(x,y,z)}{\partial(u,v)} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{bmatrix} = \begin{bmatrix} \mathbf{x}_u \\ \mathbf{x}_v \end{bmatrix}$$

1.5. Second derivatives and the Hessian Matrix

We can also take second derivatives:

$$\mathbf{H} = \frac{\partial^2(x,y,z)}{\partial(u,v)^2} = \begin{bmatrix} [x_{uu} \; y_{uu} \; z_{uu}] & [x_{uv} \; y_{uv} \; z_{uv}] \\ [x_{vu} \; y_{vu} \; z_{vu}] & [x_{vv} \; y_{vv} \; z_{vv}] \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{uu} & \mathbf{x}_{uv} \\ \mathbf{x}_{vu} & \mathbf{x}_{vv} \end{bmatrix}$$

This arrangement of second derivatives is a tensor of rank three called the *Hessian*. For the surfaces we are interested in, the skew diagonal cross derivatives are equal, so it is only necessary to store 9 numbers, not 12.

1.6. Metric Tensor or First Fundamental Form

Further derivatives of the surfaces are rarely used, however certain functions of these derivatives are used. One is the metric tensor or first fundamental form:

$$\mathbf{G} = \mathbf{T}\mathbf{T}^T = \begin{bmatrix} \mathbf{x}_u \cdot \mathbf{x}_u & \mathbf{x}_u \cdot \mathbf{x}_v \\ \mathbf{x}_v \cdot \mathbf{x}_u & \mathbf{x}_v \cdot \mathbf{x}_v \end{bmatrix}$$

which is used for measuring distances along a curve on the surface:

$$s = \int_{t_0}^{t_1} \sqrt{\dot{\mathbf{u}}\mathbf{G}\dot{\mathbf{u}}^T}\, dt$$

the differential unit of measure, $\dot{\mathbf{u}}\mathbf{G}\dot{\mathbf{u}}^T$, must be used because the (u, v) coordinates do not lie in a Euclidean space.

1.7. Curvature Tensor or the Second Fundamental Form

From the second derivatives, we can compute the curvature tensor or second fundamental form:

$$\mathbf{D} = \mathbf{n} \cdot \mathbf{H} = \begin{bmatrix} \mathbf{n} \cdot \mathbf{x}_{uu} & \mathbf{n} \cdot \mathbf{x}_{uv} \\ \mathbf{n} \cdot \mathbf{x}_{vu} & \mathbf{n} \cdot \mathbf{x}_{vv} \end{bmatrix}$$

which is used to measure normal deviation from the tangent plane in the direction $\dot{\mathbf{u}}$:

$$\dot{\mathbf{u}}\mathbf{D}\dot{\mathbf{u}}^T.$$

The normal curvature in the direction $\dot{\mathbf{u}}$ is given by:

$$\kappa_n = \frac{\dot{\mathbf{u}}\mathbf{D}\dot{\mathbf{u}}^T}{\dot{\mathbf{u}}\mathbf{G}\dot{\mathbf{u}}^T}.$$

1.8. Gaussian Curvature

The Gaussian curvature is given by:

$$K = \frac{|\mathbf{D}|}{|\mathbf{G}|}$$

1.9. Normal Curvature for Implicit Surfaces

Normal curvature can be computed for an implicit surface f(x,y,z) with [Hanrahan&Mitchell]:

$$\kappa_n = -\frac{d\mathbf{x} \cdot d\mathbf{n}}{d\mathbf{x} \cdot d\mathbf{x}}$$

where d**x** is a differential direction in the tangent plane, and d**n** is the differential normal, given by:

$$d\mathbf{n} = \frac{\left((\mathbf{g} \cdot \mathbf{g})\mathbf{I} - (\mathbf{g}\mathbf{g}^T)\right)d\mathbf{g}}{(\mathbf{g} \cdot \mathbf{g})^{\frac{3}{2}}}$$

where **g** is the gradient of f (**g** = ∇f) whose differential is:

$$d\mathbf{g} = \mathbf{H}d\mathbf{x} = \begin{bmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{yx} & f_{yy} & f_{yz} \\ f_{zx} & f_{zy} & f_{zz} \end{bmatrix} d\mathbf{x}$$

1.10. Texture-to-Screen Jacobian Matrix

In 3D computer graphics, we visualize geometric models by projection through a camera onto a pixel grid on the screen. The jacobian matrix of the coordinates in parametric space (u,v) to the coordinates in the pixel grid (i,j) gives us an idea of the shape of the pixel as projected onto the parametric surface.

$$\mathbf{J} = \frac{\partial(u,v)}{\partial(i,j)} = \begin{bmatrix} \frac{\partial u}{\partial i} & \frac{\partial v}{\partial i} \\ \frac{\partial u}{\partial j} & \frac{\partial v}{\partial j} \end{bmatrix} = \begin{bmatrix} u_i & v_i \\ u_j & v_j \end{bmatrix}$$

The Jacobian matrix maps the square shape of a screen "pixel" into a parallelogram. In order to adequately represent the texture image on the screen, all of the texture pixels within this parallelogram need to be taken into account when calculating the value of the screen pixel. In the general case, the Point-Spread Function (PSF) to be used for filtering will extend slightly outside this area to provide continuity in color from one pixel to the next.

For a pixel of unit radius, the function

$$f(\theta) = [\cos\theta \quad \sin\theta]\mathbf{J}$$

traces out an ellipse in the parametric space corresponding to the projection of the pixel onto the surface.

A square pixel transforms into a parallelogram in texture space:

$$\mathbf{Q} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix}\mathbf{J}$$

where **Q** is the parallelogram in texture space.

1.11. Projected Pixel as a Texture Rectangle

While this jacobian matrix itself can be useful for anti-aliasing texture-maps [Heckbert], other texture-mapping algorithms use derivations of this. The summed-area technique [Crow] approximates this ellipse with a rectangle, and the MipMap technique approximates this with a circle. The rectangle is computed as:

$$\Delta u = |u_i| + |u_j|$$

$$\Delta v = |v_i| + |v_j|$$

1.12. Projected Pixel as a Texture Circle (or Square)

and the diameter of the circle is computed as:

$$d = \|\mathbf{J}\|$$

for some suitable norm. Some of the commonly used norms are given below and evaluated in the appendix:

$$\left\|\left[a_{ij}\right]\right\|_1 = \max_i \sum_j |a_{ij}|$$

$$\left\|\left[a_{ij}\right]\right\|_2 = \max_i |\lambda_i|; \quad \left\|\left[a_{ij}\right]_{2x2}\right\|_2 = \left|\frac{|a_{00} + a_{11}| + \sqrt{(a_{00} + a_{11})^2 - 4(a_{00}a_{11} - a_{01}a_{10})}}{2}\right|$$

$$\left\|\left[a_{ij}\right]\right\|_{Heckbert} = \max_i \sqrt{\sum_j a_{ij}^2}$$

$$\left\|\left[a_{ij}\right]\right\|_\infty = \max_j \sum_i |a_{ij}|$$

$$\sqrt{\left\|\left[a_{ij}\right]\left[a_{ij}\right]^T\right\|_2} = \sqrt{\frac{a_{00}^2 + a_{01}^2 + a_{10}^2 + a_{11}^2 + \sqrt{(a_{00}^2 + a_{01}^2 + a_{10}^2 + a_{11}^2)^2 - 4(a_{00}a_{11} - a_{01}a_{10})^2}}{2}}$$

1.13. 3D Parametrization

To facilitate placement of solid textures, it is useful to have a 3D parametrization different from that of the world space coordinates. Often, this is the modeling (local) space coordinates, but it may be different for æsthetic reasons. In most cases, this parametrization is an affine transform of the world space coordinates:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \begin{bmatrix} u & v & w & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \\ x_o & y_o & z_o \end{bmatrix}$$

where (u,v,w) is the 3D parametrization.

1.14. Tangents of 3D Parametrization

We can define tangents with respect to the 3D parametrization as we did with 2D parametrizations. If the relation with world space coordinates is an affine one, then the tangents are simply the top three rows of the affine mapping matrix:

$$\frac{\partial(x,y,z)}{\partial(u,v,w)} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}$$

1.15. 3D-Texture-to-Screen Jacobian Matrix
For anti-aliasing purposes, it is useful to have the jacobian matrix of the 3D parametrization to screen space:

$$\frac{\partial(u,v,w)}{\partial(i,j)} = \begin{bmatrix} u_i & v_i & w_i \\ u_j & v_j & w_j \end{bmatrix}$$

This maps a circular pixel in screen space into an ellipse in parametric 3-space, or a square pixel in screen space to a parallelogram in parametric 3-space. This can be used to filter a texture anisotropically, to get the maximum sharpness in each direction.

1.16. Projected Pixel as a 3D Texture Sphere
When generating an anti-aliased solid texture, it is sometimes more convenient to filter the texture isotropically. For this, we would like to reduce the above jacobian matrix to one number. For this, the L_1, L_∞ or L_{Heckbert} norms (as described above) are appropriate.

1.17. Ray
There is a ray that comes from the camera through the given position in space. With a projective camera, all rays pass through the center of projection. With an orthographic or oblique camera, no rays pass through the same point (except for the point at infinity).

1.18. The Geometric Neighborhood Data Structure
We can bundle all of these into a C structure:

```
struct SurfaceNeighborhood {
   Point3D     position;         /* x,y,z */
   Vector3D    normal;           /* nx, ny, nz */
   Point2D     parametrization;  /* u,v */
   Vector3D    tangent[2];       /* ∂(x,y,z)/∂u, ∂(x,y,z)/∂v */
   Vector3D    hessian[3];       /* uu, uv, vv */
   Matrix2x2   metric;
   Matrix2x2   curvature;
   float       gaussianCurvature;
   Matrix2x2   paramScreenJacobian;
   Vector2D    paramScreenRect;
   float       paramScreenCircle;
   Point3D     parametrization3D;
   Vector3D    tangent3D[3];
   Matrix2x3   param3DScreenJacobian;
   float       param3DScreenSphere;
};
```

We have given methods for computing all of the quantities in this structure except for the jacobian of parametrization to screen. First, we need to know something about the camera transformation.

2. Computing 2D Parametric Tangents for Polygons
Given the (x,y,z) and (u,v) coordinates for three consecutive points:

$$\begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix} \leftrightarrow \begin{bmatrix} u_0 & v_0 \end{bmatrix}$$
$$\begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix} \leftrightarrow \begin{bmatrix} u_1 & v_1 \end{bmatrix}$$
$$\begin{bmatrix} x_2 & y_2 & z_2 \end{bmatrix} \leftrightarrow \begin{bmatrix} u_2 & v_2 \end{bmatrix}$$

the unique affine mapping that is consistent with this mapping is given by:

$$\begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix} = \begin{bmatrix} u_0 & v_0 & 1 \\ u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_o & y_o & z_o \end{bmatrix}$$

The right matrix specifies an affine transform from (u,v) to (x,y,z), and may be solved for with Gaussian elimination or L-U decomposition; this is recommended over inversion of the (u,v) matrix because it is faster and more accurate. The top two rows of this matrix are the tangents with respect to u and v.

We can optimize this a bit more, since we are only looking for the tangents, not the entire transformation function. If we subtract the last row from the first and second rows in the (x,y,z) and (u,v) matrices, we get:

$$\begin{bmatrix} \Delta x_0 & \Delta y_0 & \Delta z_0 \\ \Delta x_1 & \Delta y_1 & \Delta z_1 \end{bmatrix} = \begin{bmatrix} \Delta u_0 & \Delta v_0 \\ \Delta u_1 & \Delta v_1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{bmatrix}$$

and after inversion, the tangents are given by:

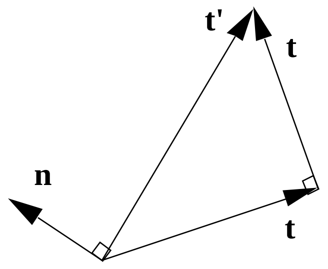
$$\begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{bmatrix} = \begin{bmatrix} \Delta u_0 & \Delta v_0 \\ \Delta u_1 & \Delta v_1 \end{bmatrix}^{-1} \begin{bmatrix} \Delta x_0 & \Delta y_0 & \Delta z_0 \\ \Delta x_1 & \Delta y_1 & \Delta z_1 \end{bmatrix}$$

This requires less than half the amount of computation, since inversion or LU decomposition of a 2x2 matrix is less than half the work of a 3x3 matrix.

This works fine for triangles, but it is sometimes desired to estimate tangents for a polygon with more than 3 sides. We suggest repeating this computation for all sets of 3 consecutive points. This results in tangents that vary over the surface of the polygon.

Sometimes, polygons are used to represent a tessellated curved surface. In this case, the normals are probably sampled at the vertices as well. Alternatively, there are algorithms that estimate normals from incident faces.

The computed tangents can be corrected to be consistent with the normal by assuring that the component of the corrected tangent in the direction of the initial tangent is equal to the initial tangent. This is illustrated in the diagram below:



$$\mathbf{t}' = \frac{|\mathbf{t}|^2}{|\mathbf{n} \times \mathbf{t}|^2} (\mathbf{n} \times \mathbf{t}) \times \mathbf{n}$$

The advantage to this kind of projection is that the corrected tangent has a component that is equal to tangent of the polygon. Another vertex would also have this property, so that when the tangents are interpolated, only the $\mathbf{t}$ components would be interpolated.

3. Computing Curvature for Polygons

Polygons are flat by their nature, so they have no curvature. But if the polygon was created from sampling points and normals from a curved surface, the normal curvature may be estimated from the rate of change of the unit tangents.

4. Camera Model for Projecting Points

The viewing pipeline is assumed to take the following form:

$$\mathbf{q} = \mathbf{pMVCS}$$

where

$$\mathbf{q}_{1\times 4} = \mathbf{p}_{1\times 4}\mathbf{M}_{4\times 4}\mathbf{V}_{4\times 4}\mathbf{C}_{4\times 4}\mathbf{S}_{4\times 4}$$

specifies the sizes of the matrices involved

$$\mathbf{p} = \begin{bmatrix} x_m & y_m & z_m & 1 \end{bmatrix}$$

is a point in modeling (local) space, and

$$\mathbf{q} = \begin{bmatrix} x_s & y_s & z_s & w_s \end{bmatrix}$$

is a point in screen space. The matrices:

$$\mathbf{M} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & 0 \\ m_{10} & m_{11} & m_{12} & 0 \\ m_{20} & m_{21} & m_{22} & 0 \\ m_{30} & m_{31} & m_{32} & 1 \end{bmatrix}$$

maps model to world space, and takes on general values,

$$\mathbf{V} = \begin{bmatrix} v_{00} & v_{01} & v_{02} & 0 \\ v_{10} & v_{11} & v_{12} & 0 \\ v_{20} & v_{21} & v_{22} & 0 \\ v_{30} & v_{31} & v_{32} & 1 \end{bmatrix}$$

maps world to camera space, and is orthonormal in the upper left 3x3,

$$\mathbf{C} = \begin{bmatrix} c_{00} & 0 & 0 & 0 \\ 0 & c_{11} & 0 & 0 \\ c_{20} & c_{21} & c_{22} & c_{23} \\ 0 & 0 & 0 & c_{33} \end{bmatrix}$$

maps camera to clipping space, and

$$\mathbf{S} = \begin{bmatrix} s_{00} & 0 & 0 & 0 \\ 0 & s_{11} & 0 & 0 \\ 0 & 0 & s_{22} & 0 \\ s_{30} & s_{31} & 0 & 1 \end{bmatrix}$$

maps clipping to screen space.

Let us call the concatenation of the matrices **A**:

$$\mathbf{A} = \mathbf{MVCS}$$

where

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Then

$$\mathbf{q} = \mathbf{pA}$$

maps from model space to screen space.

5. Determining the Ray

If **A** maps from world (or modeling) space to screen space, then $\mathbf{A}^{-1}$ maps from screen space to world space:

$$\begin{bmatrix} xw & yw & zw & w \end{bmatrix} = \begin{bmatrix} i & j & k & 1 \end{bmatrix} \mathbf{A}^{-1}$$

or

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} i & j & k & 1 \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \\ b_{30} & b_{31} & b_{32} \end{bmatrix}}{\begin{bmatrix} i & j & k & 1 \end{bmatrix} \begin{bmatrix} b_{03} \\ b_{13} \\ b_{23} \\ b_{33} \end{bmatrix}}$$

where

$$\mathbf{B} = \mathbf{A}^{-1}.$$

This transforms from screen space to world space. The plane at k=0 corresponds to the near clipping plane, and the plane at k=-∞ corresponds to the eye point[1].

In this section, we find it useful to define the intermediate variables:

$$\begin{bmatrix} c_x & c_y & c_z & c_w \end{bmatrix} = \begin{bmatrix} i & j & 1 \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

This then reduces the above transformation to:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} b_{20}k + c_x & b_{21}k + c_y & b_{22}k + c_z \end{bmatrix}}{b_{23}k + c_w}$$

5.1. Ray from Screen Point #1

Given a pixel (i, j), we find that the ray is given by:

[1]This is the result in systems that have the depth variable, k, pointing away from the viewer. In systems that have k pointing toward the viewer, the eye point corresponds to k=+∞.

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} b_{20}k + c_x & b_{21}k + c_y & b_{22}k + c_z \end{bmatrix}}{b_{23}k + c_w}$$

However, it can also be represented in the form:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \begin{bmatrix} a_x t + b_x & a_y t + b_y & a_z t + b_t \end{bmatrix}$$

by a suitable case analysis.

Case 1: $b_{23} \neq 0$ and $c_w \neq 0$.
Two points on the ray can be found by substituting k = -∞ (eye point) and k = 0 (near clipping plane):

$$\begin{bmatrix} x_{-\infty} & y_{-\infty} & z_{-\infty} \end{bmatrix} = \begin{bmatrix} \frac{b_{20}}{b_{23}} & \frac{b_{21}}{b_{23}} & \frac{b_{22}}{b_{23}} \end{bmatrix}$$

$$\begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix} = \begin{bmatrix} \frac{c_x}{c_w} & \frac{c_y}{c_w} & \frac{c_z}{c_w} \end{bmatrix}$$

yielding the ray emanating from the eyepoint:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix}}{b_{23}} + t\left\{ \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w} - \frac{\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix}}{b_{23}} \right\}$$

or by reparametrizing t:, a more robust version:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix}}{b_{23}} + t\left(\begin{bmatrix} c_x & c_y & c_z \end{bmatrix} - c_w \frac{\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix}}{b_{23}} \right)$$

Alternatively, we can generate a ray starting at the near clipping plane:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w} + t\left(\frac{\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix}}{b_{23}} - \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w} \right)$$

or a more robust version:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w} + t\left(\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix} - b_{23} \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w} \right)$$

This corresponds to the normal perspective camera.

Case 2: $b_{23} = 0$ and $c_w \neq 0$.
Simple division results in:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix} + k\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix}}{c_w}$$

With the parametrization:

$$t = \frac{k}{c_w}$$

we get:

$$[x \quad y \quad z] = \frac{[c_x \quad c_y \quad c_z]}{c_w} + t[b_{20} \quad b_{21} \quad b_{22}]$$

This case corresponds to a non-perspective (orthographic, oblique) camera. Since the eye point of a non-perspective camera is at -∞, this ray starts at the near clipping plane.

Case 3: $b_{23} \neq 0$ and $c_w = 0$.
We find two points at $k = -\infty$ and $k = 1$:

$$[x_{-\infty} \quad y_{-\infty} \quad z_{-\infty}] = \begin{bmatrix} \frac{b_{20}}{b_{23}} & \frac{b_{21}}{b_{23}} & \frac{b_{22}}{b_{23}} \end{bmatrix}$$

$$[x_1 \quad y_1 \quad z_1] = \begin{bmatrix} \frac{b_{20}+c_x}{b_{23}} & \frac{b_{21}+c_y}{b_{23}} & \frac{b_{22}+c_z}{b_{23}} \end{bmatrix}$$

yielding the ray:

$$[x \quad y \quad z] = \frac{[b_{20} \quad b_{21} \quad b_{22}]}{b_{23}} + t\frac{[c_x \quad c_y \quad c_z]}{b_{23}}$$

This does not correspond to any type of camera we normally have in computer graphics, because it blows up at k=0.

5.2. Ray from Screen Point #2

We reparametrize k by the Möbius parametrization:

$$t = \frac{1}{b_{23}k + c_w}$$

and find that:

$$k = \frac{1 - c_w t}{b_{23} t}$$

This then yields:

$$[x \quad y \quad z] = \frac{[b_{20} \quad b_{21} \quad b_{22}]}{b_{23}} + t\left([c_x \quad c_y \quad c_z] - c_w \frac{[b_{20} \quad b_{21} \quad b_{22}]}{b_{23}}\right)$$

5.3. Ray from Screen Point #3

With the Möbius parametrization:

$$t = \frac{k}{b_{23}k + c_w}$$

gives

$$k = \frac{c_w t}{1 - b_{23} t}$$

and

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w} + t\left(\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix} - b_{23} \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w} \right)$$

5.4. Ray from World Space Point

Again, we use the $\mathbf{A}^{-1}$ matrix, where $\mathbf{A}^{-1}$ maps from screen space to world space:

$$\begin{bmatrix} xw & yw & zw & w \end{bmatrix} = \begin{bmatrix} i & j & k & 1 \end{bmatrix} \mathbf{A}^{-1}$$

Here, we find that the world space points are given by:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \frac{\begin{bmatrix} b_{00} i + b_{10} j + b_{20} k + b_{30} & b_{01} i + b_{11} j + b_{21} k + b_{31} & b_{02} i + b_{12} j + b_{22} k + b_{32} \end{bmatrix}}{b_{03} i + b_{13} j + b_{23} k + b_{33}}$$

where

$$\mathbf{B} = \mathbf{A}^{-1}.$$

We already have a point on the ray; the direction may be found by taking the tangent with respect to *k*:

$$\frac{\partial (x, y, z)}{\partial k} = \frac{1}{w} \begin{bmatrix} b_{20} - b_{23} x & b_{21} - b_{23} y & b_{22} - b_{23} z \end{bmatrix}$$

where the b_{ij}'s are the elements of the $\mathbf{A}^{-1}$ matrix, *w* is the [common] denominator, and we have made use of the quotient rule for derivatives.

Scaling by the common denominator (*w*), we arrive at:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix} + t\left(\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix} - b_{23} \begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix} \right)$$

where (x_0, y_0, z_0) is the given world space point.

6. Determining the Ray Wavefront

The ray wavefront is an enhancement of the ray, in that it also carries with it vectors corresponding to a unit step in screen *x* and *y* as well as curvature of the ray wavefront that indicate how the ray grows with distance.

Given the ray:

$$\begin{bmatrix} x_0 & x_1 & x_2 \end{bmatrix} = \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w} + t\left(\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix} - b_{23} \frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w} \right)$$

its derivatives with respect to screen coordinates are:

$$\frac{\partial(x,y,z)}{\partial(i,j)} = \frac{(1-b_{23}t)}{c_w}\left\{\begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{bmatrix} - \begin{bmatrix} b_{03} \\ b_{13} \end{bmatrix}\frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w}\right\}$$

At the near clipping plane (*t*=0), the derivatives are:

$$\mathbf{J} = \frac{\begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{bmatrix} - \begin{bmatrix} b_{03} \\ b_{13} \end{bmatrix}\frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w}}{c_w}$$

The ray normal (i.e. direction) is:

$$\mathbf{g} = \begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix} - b_{23}\frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w}$$

and its normalized direction is computed as:

$$\mathbf{n} = \frac{\mathbf{g}}{|\mathbf{g}|}$$

The derivatives of this ray direction with respect to the pixel coordinates are:

$$\frac{\partial\mathbf{n}}{\partial(i,j)} = \frac{|\mathbf{g}|\frac{\partial\mathbf{g}}{\partial(i,j)} - \frac{\partial|\mathbf{g}|}{\partial(i,j)}\mathbf{g}}{|\mathbf{g}|^2} = \frac{\frac{\partial\mathbf{g}}{\partial(i,j)} - \frac{\partial|\mathbf{g}|}{\partial(i,j)}\mathbf{n}}{|\mathbf{g}|}$$

The derivative of the numerator is:

$$\frac{\partial\mathbf{g}}{\partial(i,j)} = \frac{-b_{23}}{c_w}\left\{\begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{bmatrix} - \begin{bmatrix} b_{03} \\ b_{13} \end{bmatrix}\frac{\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}}{c_w}\right\} = -b_{23}\mathbf{J}$$

and that of the denominator is:

$$\frac{\partial|\mathbf{g}|}{\partial(i,j)} = \frac{\partial}{\partial(i,j)}\left(\mathbf{g}\mathbf{g}^T\right)^{\frac{1}{2}} = \frac{1}{2}\left(\mathbf{g}\mathbf{g}^T\right)^{-\frac{1}{2}}\frac{\partial}{\partial(i,j)}\left(\mathbf{g}\mathbf{g}^T\right) = \frac{1}{2|\mathbf{g}|}\frac{\partial}{\partial(i,j)}\left(\mathbf{g}\mathbf{g}^T\right)$$

where

$$\frac{\partial}{\partial(i,j)}\left(\mathbf{g}\mathbf{g}^T\right) = \mathbf{g}\frac{\partial\mathbf{g}^T}{\partial(i,j)} + \frac{\partial\mathbf{g}}{\partial(i,j)}\mathbf{g}^T = 2\frac{\partial\mathbf{g}}{\partial(i,j)}\mathbf{g}^T = -2b_{23}\mathbf{J}\mathbf{g}^T$$

so that the derivative of the denominator is:

$$\frac{\partial|\mathbf{g}|}{\partial(i,j)} = \frac{1}{2|\mathbf{g}|}\left(-2b_{23}\mathbf{J}\mathbf{g}^T\right) = -b_{23}\mathbf{J}\mathbf{n}^T$$

and the desired derivatives of the normal is:

$$\frac{\partial\mathbf{n}}{\partial(i,j)} = \frac{-b_{23}\mathbf{J} - \left(-b_{23}\mathbf{J}\mathbf{n}^T\right)\mathbf{n}}{|\mathbf{g}|} = \frac{b_{23}}{|\mathbf{g}|}\left(\mathbf{J}\mathbf{n}^T\mathbf{n} - \mathbf{J}\right)$$

Curvature is given as:

$$\kappa = -\frac{d\mathbf{n}\bullet d\mathbf{t}}{d\mathbf{t}\bullet d\mathbf{t}}$$

Yielding:

$$
\begin{aligned}
\kappa &= \frac{b_{23}}{|\mathbf{g}|} \frac{\left\{ \mathbf{J} - \mathbf{J}\mathbf{n}^T\mathbf{n} \right\} \bullet \mathbf{J}}{\mathbf{J} \bullet \mathbf{J}} \\
&= \frac{b_{23}}{|\mathbf{g}|} \frac{\mathbf{J} \bullet \mathbf{J} - \mathbf{J}\mathbf{n}^T\mathbf{n} \bullet \mathbf{J}^T}{\mathbf{J} \bullet \mathbf{J}} \\
&= \frac{b_{23}}{|\mathbf{g}|} \left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \frac{\mathbf{J}\mathbf{n}^T \bullet \left(\mathbf{J}\mathbf{n}^T \right)}{\mathbf{J} \bullet \mathbf{J}} \right\}
\end{aligned}
$$

where the dot products are taken separately for *i* and *j* (a slight abuse of notation), resulting in:

$$
\kappa_i = \frac{b_{23}}{|\mathbf{g}|} \left\{ 1 - \frac{\left(\mathbf{J}_i \bullet \mathbf{n} \right)^2}{\mathbf{J}_i \bullet \mathbf{J}_i} \right\} \qquad \kappa_j = \frac{b_{23}}{|\mathbf{g}|} \left\{ 1 - \frac{\left(\mathbf{J}_j \bullet \mathbf{n} \right)^2}{\mathbf{J}_j \bullet \mathbf{J}_j} \right\}
$$

7. Determining the Jacobian of 2D Parametrization to Screen Space

7.1. Jacobian by the Method of Tangents

To perform anti-aliased texture-mapping, we seek the jacobian of parametrization-to-screen space:

$$\frac{\partial(u,v)}{\partial(i,j)}$$

Let us represent the projective matrix **A** by the function *a*(•), which maps points from world space to screen space:

$$a : \mathbf{R}^3 \to \mathbf{R}^2$$

or

$$a : (x, y, z) \mapsto (i, j)$$

We will represent the parametric surface by b(•),

$$b : \mathbf{R}^2 \to \mathbf{R}^3$$

mapping

$$b : (u,v) \mapsto (x, y, z)$$

We first construct the composition function c(•)

$$c : \mathbf{R}^2 \to \mathbf{R}^2$$

mapping

$$c : (u,v) \mapsto (i, j)$$

as such:

$$c = a \circ b$$

or

$$c(u,v) = a(b(u,v))$$

From this, we evaluate the jacobian:

$$\frac{\partial c}{\partial(u,v)} = \frac{\partial(i,j)}{\partial(u,v)}$$

and invert this 2x2 matrix to yield the desired jacobian:

$$\frac{\partial(u,v)}{\partial(i,j)} = \left[\frac{\partial(i,j)}{\partial(u,v)}\right]^{-1}$$

The quantity

$$\frac{\partial(i,j)}{\partial(u,v)}$$

can be computed as the product of jacobians:

$$\frac{\partial(i,j)}{\partial(u,v)} = \left.\frac{\partial(x,y,z)}{\partial(u,v)}\right|_{(x_0,y_0,z_0)} \left.\frac{\partial(i,j)}{\partial(x,y,z)}\right|_{(x_0,y_0,z_0)}$$

The former is trivial to compute, since it is the jacobian of tangents. The function *a(•)*, which maps modeling space to world space, is a simple projective transformation:

$$i = \frac{a_{00}x + a_{10}y + a_{20}z + a_{30}}{a_{03}x + a_{13}y + a_{23}z + a_{33}} = \frac{p_0(x,y,z)}{p_3(x,y,z)}$$

$$j = \frac{a_{01}x + a_{11}y + a_{21}z + a_{31}}{a_{03}x + a_{13}y + a_{23}z + a_{33}} = \frac{p_1(x,y,z)}{p_3(x,y,z)},$$

so its derivatives are computed by the quotient rule, e.g.

$$\frac{\partial i}{\partial x} = \frac{p_3\frac{\partial p_0}{\partial x} - p_0\frac{\partial p_3}{\partial x}}{p_3^2} = \frac{\frac{\partial p_0}{\partial x} - i\frac{\partial p_3}{\partial x}}{p_3} = \frac{a_{00} - ia_{03}}{p_3}$$

etc., yielding the jacobian:

$$\frac{\partial(i,j)}{\partial(x,y,z)} = \frac{1}{p_3}\begin{bmatrix} a_{00} - ia_{03} & a_{01} - ja_{03} \\ a_{10} - ia_{13} & a_{11} - ja_{13} \\ a_{20} - ia_{23} & a_{21} - ja_{23} \end{bmatrix}$$

This is then multiplied by the tangent jacobian,

$$\frac{\partial(x,y,z)}{\partial(u,v)} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{bmatrix}$$

to yield a 2x2 matrix that is the inverse of the desired jacobian:

$$\frac{\partial(i,j)}{\partial(u,v)} = \frac{1}{p_3}\begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{bmatrix}\begin{bmatrix} a_{00} - ia_{03} & a_{01} - ja_{03} \\ a_{10} - ia_{13} & a_{11} - ja_{13} \\ a_{20} - ia_{23} & a_{21} - ja_{23} \end{bmatrix}$$

By inverting this, we find the desired result as:

$$\frac{\partial(u,v)}{\partial(i,j)} = p_3\left\{\begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{bmatrix}\begin{bmatrix} a_{00} - ia_{03} & a_{01} - ja_{03} \\ a_{10} - ia_{13} & a_{11} - ja_{13} \\ a_{20} - ia_{23} & a_{21} - ja_{23} \end{bmatrix}\right\}^{-1}$$

7.1.1. Separation of the Screen Map

Suppose that the screen mapping is separated from the rest of the transformation, or maybe an additional screen mapping is applied. In this case, we apply another function:

$d \circ a \circ b$

where *d(•)* is the new screen mapping. This can be represented by a matrix similar to **S**:

$$\mathbf{D} = \begin{bmatrix} d_{00} & 0 & 0 & 0 \\ 0 & d_{11} & 0 & 0 \\ 0 & 0 & d_{22} & 0 \\ d_{30} & d_{31} & 0 & 1 \end{bmatrix}$$

The concatenation of jacobians is:

$$\frac{\partial(i,j)}{\partial(u,v)} = \left.\frac{\partial(x,y,z)}{\partial(u,v)}\right|_{(x_0,y_0,z_0)} \left.\frac{\partial(k,l)}{\partial(x,y,z)}\right|_{(x_0,y_0,z_0)} \frac{\partial(i,j)}{\partial(k,l)}$$

where (*k*,*l*) is an intermediate 2-D coordinate system.

The last screen jacobian is:

$$\frac{\partial(i,j)}{\partial(k,l)} = \begin{bmatrix} d_{00} & 0 \\ 0 & d_{11} \end{bmatrix}$$

and its inverse is:

$$\frac{\partial(k,l)}{\partial(i,j)} = \left[\frac{\partial(i,j)}{\partial(k,l)}\right]^{-1} = \begin{bmatrix} \frac{1}{d_{00}} & 0 \\ 0 & \frac{1}{d_{11}} \end{bmatrix}$$

Computing the desired jacobian, we have:

$$\frac{\partial(u,v)}{\partial(i,j)} = \left[\frac{\partial(i,j)}{\partial(u,v)}\right]^{-1}$$

$$= \left[\left.\frac{\partial(k,l)}{\partial(u,v)}\right|_{(x_0,y_0,z_0)} \frac{\partial(i,j)}{\partial(k,l)}\right]^{-1}$$

$$= \left[\frac{\partial(i,j)}{\partial(k,l)}\right]^{-1} \left[\left.\frac{\partial(k,l)}{\partial(u,v)}\right|_{(x_0,y_0,z_0)}\right]^{-1}$$

$$= \begin{bmatrix} \frac{1}{d_{00}} & 0 \\ 0 & \frac{1}{d_{11}} \end{bmatrix} \frac{\partial(u,v)}{\partial(k,l)}$$

i.e. the screen mapping affects the jacobian only in the sense that the input is scaled by the pixel resolution in each direction.

Therefore, the complete computations for the texture-to-screen jacobian, with screen transformation filtered out, is:

$$\frac{\partial(u,v)}{\partial(i,j)} = p_3 \begin{bmatrix} \frac{1}{d_{00}} & 0 \\ 0 & \frac{1}{d_{11}} \end{bmatrix} \left\{ \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{bmatrix} \begin{bmatrix} a_{00} - ka_{03} & a_{01} - la_{03} \\ a_{10} - ka_{13} & a_{11} - la_{13} \\ a_{20} - ka_{23} & a_{21} - la_{23} \end{bmatrix} \right\}^{-1}$$

Note that (*k*, *l*) (from the intermediate coordinate system, e.g. frustum space) is used in computation of the rightmost matrix, not (*i, j*) (screen space).

7.2. Jacobian from Projective Mappings

Given a projective mapping

$$\begin{bmatrix} \tilde{u} & \tilde{v} & \tilde{w} \end{bmatrix} = \begin{bmatrix} i & j & 1 \end{bmatrix} \begin{bmatrix} \tilde{u}_i & \tilde{v}_i & \tilde{w}_i \\ \tilde{u}_j & \tilde{v}_j & \tilde{w}_j \\ \tilde{u}_k & \tilde{v}_k & \tilde{w}_k \end{bmatrix}$$

mapping from screen space to projective texture space, the Euclidean coordinates may be determined from:

$$u = \frac{\tilde{u}}{\tilde{w}},\ v = \frac{\tilde{v}}{\tilde{w}}$$

The jacobian matrix of (u,v) with respect to (i,j) may be determined from the quotient rule for partial derivatives, i.e.

$$\frac{\partial u}{\partial x} = \frac{\partial}{\partial x}\left(\frac{\tilde{u}}{\tilde{w}}\right) = \frac{\tilde{w}\tilde{u}_x - \tilde{u}\tilde{w}_x}{\tilde{w}^2}$$

yielding:

$$\mathbf{J} = \frac{\partial(u,v)}{\partial(i,j)} = \frac{1}{\tilde{w}^2} \begin{bmatrix} \tilde{w}\tilde{u}_i - \tilde{u}\tilde{w}_i & \tilde{w}\tilde{v}_i - \tilde{v}\tilde{w}_i \\ \tilde{w}\tilde{u}_j - \tilde{u}\tilde{w}_j & \tilde{w}\tilde{v}_j - \tilde{v}\tilde{w}_j \end{bmatrix} = \frac{1}{\tilde{w}} \begin{bmatrix} \tilde{u}_i - u\tilde{w}_i & \tilde{v}_i - v\tilde{w}_i \\ \tilde{u}_j - u\tilde{w}_j & \tilde{v}_j - v\tilde{w}_j \end{bmatrix}$$

If the perspective is not all that extreme, it is possible to approximate this as:

$$\mathbf{J} = \frac{\partial(u,v)}{\partial(i,j)} \approx \frac{1}{\tilde{w}} \begin{bmatrix} \tilde{u}_i & \tilde{v}_i \\ \tilde{u}_j & \tilde{v}_j \end{bmatrix}$$

however, it is recommended that this not be used in general circumstances.

7.3. Jacobian from Texture-Screen Correspondences

Given a correspondence between points in texture and screen space:

$$\begin{bmatrix} u_0 & v_0 \end{bmatrix} \leftrightarrow \begin{bmatrix} i_0 & j_0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 & v_1 \end{bmatrix} \leftrightarrow \begin{bmatrix} i_1 & j_1 \end{bmatrix}$$

$$\begin{bmatrix} u_2 & v_2 \end{bmatrix} \leftrightarrow \begin{bmatrix} i_2 & j_2 \end{bmatrix}$$

we'd like to compute the texture-to-screen jacobian matrix, so that we can do anti-aliasing properly.

We could naïvely set up an affine transformation and compute the jacobian, but we know that we want to use projective interpolation instead of linear interpolation, so we can do better.

From [Heckbert-Moreton], we know that an advantageous way to interpolate texture for scan-converting is to interpolate the parameters *p*/*w* … and 1/*w* linearly instead of *p* linearly, and then divide the interpolated parameter *p*/*w* by the interpolated 1/*w* to get a projectively-interpolated *p*.

So, we perform a similar "trick" by generating a projective map for:

$$[u_0/w_0 \quad v_0/w_0 \quad 1/w_0] \leftrightarrow [i_0 \quad j_0]$$
$$[u_1/w_1 \quad v_1/w_1 \quad 1/w_1] \leftrightarrow [i_1 \quad j_1]$$
$$[u_2/w_2 \quad v_2/w_2 \quad 1/w_2] \leftrightarrow [i_2 \quad j_2]$$

where the *w*'s are the same as those used to divide (x,y,z) when converting from homogeneous to Euclidean coordinates. We use familiar linear equation solution methods to solve for the 3x3 projective mapping matrix:

$$[u/w \quad v/w \quad 1/w] = [i \quad j \quad 1]\begin{bmatrix} u_i & v_i & w_i \\ u_j & v_j & w_j \\ u_k & v_k & w_k \end{bmatrix}$$

and then use rational derivative techniques to find the jacobian of this projective transformation.
If there are more than 3 points, we suggest computing a projective mapping at each point, using its two nearest neighbors.

8. Determining the Jacobian of 3D Parametrization to Screen Space

We need to find the projection of the pixel onto the surface and measure its size.
We do this by first finding the derivatives of world space to screen space at a point in world space.
Given the mapping **A**, which takes us from world space to screen space,

$$[il \quad jl \quad kl \quad l] = [x \quad y \quad z \quad 1]\mathbf{A}$$

its inverse maps from screen space to world space

$$[xw \quad yw \quad zw \quad w] = [il \quad jl \quad kl \quad l]\mathbf{A}^{-1}$$

The derivatives we desire are

$$\frac{\partial(x,y,z)}{\partial(i,j)} = \frac{l}{w}\left\{\begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{bmatrix} - \begin{bmatrix} b_{03} \\ b_{13} \end{bmatrix}[x \quad y \quad z]\right\}$$

where

$$\mathbf{B} = \mathbf{A}^{-1}$$

If we start out with (*x*,*y*,*z*,1) and transform by **A** to yield (*il*, *jl*, *kl*, *l*), and then transform back with $\mathbf{A}^{-1}$ to (*x*,*y*,*z*,*w*), we should have *w*=1. *l* is given by:

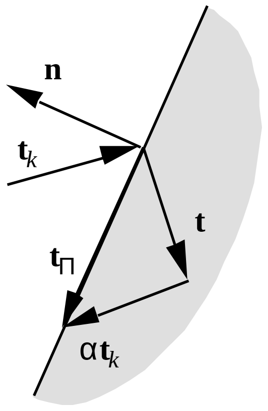
$$l = [x \quad y \quad z \quad 1]\begin{bmatrix} a_{03} \\ a_{13} \\ a_{23} \\ a_{33} \end{bmatrix}$$

This yields:

$$\frac{\partial(x,y,z)}{\partial(i,j)} = \begin{bmatrix} \mathbf{t}_i \\ \mathbf{t}_j \end{bmatrix} = \left\{ [x \quad y \quad z \quad 1]\begin{bmatrix} a_{03} \\ a_{13} \\ a_{23} \\ a_{33} \end{bmatrix}\right\}\left\{\begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{bmatrix} - \begin{bmatrix} b_{03} \\ b_{13} \end{bmatrix}[x \quad y \quad z]\right\}$$

The projection of the tangent vector $\mathbf{t}_i$ in the direction $\mathbf{t}_k$ onto the surface with normal **n** implies

$$\mathbf{t}_{i\Pi} \bullet \mathbf{n} = (\mathbf{t}_i + \alpha\mathbf{t}_k) \bullet \mathbf{n} = 0$$



so that

$$\alpha = -\frac{\mathbf{t}_i \bullet \mathbf{n}}{\mathbf{t}_k \bullet \mathbf{n}}$$

yielding

$$\mathbf{t}_{i\sqcap} = \mathbf{t}_i - \frac{\mathbf{t}_i \cdot \mathbf{n}}{\mathbf{t}_k \cdot \mathbf{n}}\mathbf{t}_k = \mathbf{t}_i\left[\tilde{\mathbf{1}} - \frac{\mathbf{n}^T\mathbf{t}_k}{\mathbf{n}\mathbf{t}_k{}^T}\right]$$

where $\tilde{\mathbf{1}}$ is the unit tensor of rank 2. The formula for the projection of $\mathbf{t}_j$ is analogous. Multiplication of the projected vectors $\mathbf{t}_i$ and $\mathbf{t}_j$ by the jacobian matrix

$$\frac{\partial(u,v,w)}{\partial(x,y,z)}$$

yields:

$$\frac{\partial(u,v,w)}{\partial(i,j)} = \begin{bmatrix}\mathbf{t}_{i\sqcap} \\ \mathbf{t}_{j\sqcap}\end{bmatrix}\frac{\partial(u,v,w)}{\partial(x,y,z)}$$

9. Determining Affine Maps

We will be primarily concerned with maps from 2-D to N-D, or 3-D to N-D, so we will specifically derive equations for these, although the methods used can be equally applied to affine maps from M-D to N-D.

For a 2-D to N-D affine map, we have:

$$\begin{bmatrix} p_1 & p_2 & \dots & p_N \end{bmatrix} = \begin{bmatrix} i & j & 1 \end{bmatrix}\begin{bmatrix} t_{1i} & t_{2i} & \dots & t_{Ni} \\ t_{1j} & t_{2j} & \dots & t_{Nj} \\ t_{1o} & t_{2o} & \dots & t_{No} \end{bmatrix}$$

where the right matrix represents the affine map. If we have 3 such correspondences between ($p_1, p_2, \dots, p_N$) and (i, j), the above relation should hold for these three simultaneously, i.e.,

$$\begin{bmatrix} p_{01} & p_{02} & \dots & p_{0N} \\ p_{11} & p_{12} & \dots & p_{1N} \\ p_{21} & p_{22} & \dots & p_{2N} \end{bmatrix} = \begin{bmatrix} i_0 & j_0 & 1 \\ i_1 & j_1 & 1 \\ i_2 & j_2 & 1 \end{bmatrix}\begin{bmatrix} t_{1i} & t_{2i} & \dots & t_{Ni} \\ t_{1j} & t_{2j} & \dots & t_{Nj} \\ t_{1o} & t_{2o} & \dots & t_{No} \end{bmatrix}$$

The affine mapping matrix may then be solved for by inverting the (i,j) matrix:

$$\begin{bmatrix} t_{1i} & t_{2i} & \cdots & t_{Ni} \\ t_{1j} & t_{2j} & \cdots & t_{Nj} \\ t_{1o} & t_{2o} & \cdots & t_{No} \end{bmatrix} = \begin{bmatrix} i_0 & j_0 & 1 \\ i_1 & j_1 & 1 \\ i_2 & j_2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} p_{01} & p_{02} & \cdots & p_{0N} \\ p_{11} & p_{12} & \cdots & p_{1N} \\ p_{21} & p_{22} & \cdots & p_{2N} \end{bmatrix}$$

For the 3-D to N-D map, the corresponding solution is:

$$\begin{bmatrix} t_{1i} & t_{2i} & \cdots & t_{Ni} \\ t_{1j} & t_{2j} & \cdots & t_{Nj} \\ t_{1k} & t_{2k} & \cdots & t_{Nk} \\ t_{1o} & t_{2o} & \cdots & t_{No} \end{bmatrix} = \begin{bmatrix} i_0 & j_0 & k_0 & 1 \\ i_1 & j_1 & k_1 & 1 \\ i_2 & j_2 & k_2 & 1 \\ i_3 & j_3 & k_3 & 1 \end{bmatrix}^{-1} \begin{bmatrix} p_{01} & p_{02} & \cdots & p_{0N} \\ p_{11} & p_{12} & \cdots & p_{1N} \\ p_{21} & p_{22} & \cdots & p_{2N} \\ p_{31} & p_{32} & \cdots & p_{3N} \end{bmatrix}$$

which relates (i,j,k) to $(p_1, p_2, \ldots, p_N)$ as follows:

$$\begin{bmatrix} p_1 & p_2 & \ldots & p_N \end{bmatrix} = \begin{bmatrix} i & j & k & 1 \end{bmatrix} \begin{bmatrix} t_{1i} & t_{2i} & \cdots & t_{Ni} \\ t_{1j} & t_{2j} & \cdots & t_{Nj} \\ t_{1k} & t_{2k} & \cdots & t_{Nk} \\ t_{1o} & t_{2o} & \cdots & t_{No} \end{bmatrix}$$

9.1. Determining the Jacobian Matrix of an Affine Map

The jacobian matrix of an affine map is a linear map that can be picked from the affine map's matrix representation by eliminating the latter's last row. However, it is not necessary to compute the entire affine transformation if only a subset (the jacobian matrix) is desired.

For 2-D, we have:

$$\frac{\partial(p_1, p_2, \ldots, p_N)}{\partial(i,j)} = \begin{bmatrix} t_{1i} & t_{2i} & \cdots & t_{Ni} \\ t_{1j} & t_{2j} & \cdots & t_{Nj} \end{bmatrix} = \begin{bmatrix} \Delta i_0 & \Delta j_0 \\ \Delta i_1 & \Delta j_1 \end{bmatrix}^{-1} \begin{bmatrix} \Delta p_{01} & \Delta p_{02} & \cdots & \Delta p_{0N} \\ \Delta p_{11} & \Delta p_{12} & \cdots & \Delta p_{1N} \end{bmatrix}$$

and for 3-D we have:

$$\frac{\partial(p_1, p_2, \ldots, p_N)}{\partial(i,j,k)} = \begin{bmatrix} t_{1i} & t_{2i} & \cdots & t_{Ni} \\ t_{1j} & t_{2j} & \cdots & t_{Nj} \\ t_{1k} & t_{2k} & \cdots & t_{Nk} \end{bmatrix} = \begin{bmatrix} \Delta i_0 & \Delta j_0 & \Delta k_0 \\ \Delta i_1 & \Delta j_1 & \Delta k_1 \\ \Delta i_2 & \Delta j_2 & \Delta k_2 \end{bmatrix}^{-1} \begin{bmatrix} \Delta p_{01} & \Delta p_{02} & \cdots & \Delta p_{0N} \\ \Delta p_{11} & \Delta p_{12} & \cdots & \Delta p_{1N} \\ \Delta p_{21} & \Delta p_{22} & \cdots & \Delta p_{2N} \end{bmatrix}$$

These equations use the inverse of a matrix of order one less than that required for an affine map.

10. Determining Projective Mappings

10.1. 2D-2D Projective Mapping from Screen Space Correspondence

From [Heckbert89a] , we can compute the projective mapping between a unit square and an arbitrary quadrilateral in (x,y). This is given by the projective frame:

$$\mathbf{R}_{projective} = \begin{bmatrix} x_1 - x_0 + gx_1 & y_1 - y_0 + gy_1 & g \\ x_3 - x_0 + hx_3 & y_3 - y_0 + hy_3 & h \\ x_0 & y_0 & 1 \end{bmatrix}$$

where

[Heckbert89a] Heckbert, Paul, Fundamentals of Texture Mapping and Image Warping, master's thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1989.

$$g = \frac{\begin{vmatrix} \sum x & \Delta x_{32} \\ \sum y & \Delta y_{32} \end{vmatrix}}{\begin{vmatrix} \Delta x_{12} & \Delta x_{32} \\ \Delta y_{12} & \Delta y_{32} \end{vmatrix}}, \qquad h = \frac{\begin{vmatrix} \Delta x_{12} & \sum x \\ \Delta y_{12} & \sum y \end{vmatrix}}{\begin{vmatrix} \Delta x_{12} & \Delta x_{32} \\ \Delta y_{12} & \Delta y_{32} \end{vmatrix}}$$

and

$$\Delta x_{12} = x_1 - x_2,\ \Delta x_{32} = x_3 - x_2, \qquad \sum x = x_0 - x_1 + x_2 - x_3,$$

$$\Delta y_{12} = y_1 - y_2,\ \Delta y_{32} = y_3 - y_2, \qquad \sum y = y_0 - y_1 + y_2 - y_3,$$

Alternatively, we note that the computations for g and h resemble the ratio of two cross products:

$$g = \frac{\Delta_{01} \times \Delta_{32}}{\Delta_{12} \times \Delta_{32}}, \qquad h = \frac{\Delta_{12} \times \Delta_{03}}{\Delta_{12} \times \Delta_{32}}$$

where

$$\Delta_{01} = p_0 - p_1,\ \ \Delta_{03} = p_0 - p_3,\ \ \Delta_{12} = p_1 - p_2,\ \ \Delta_{32} = p_3 - p_2$$

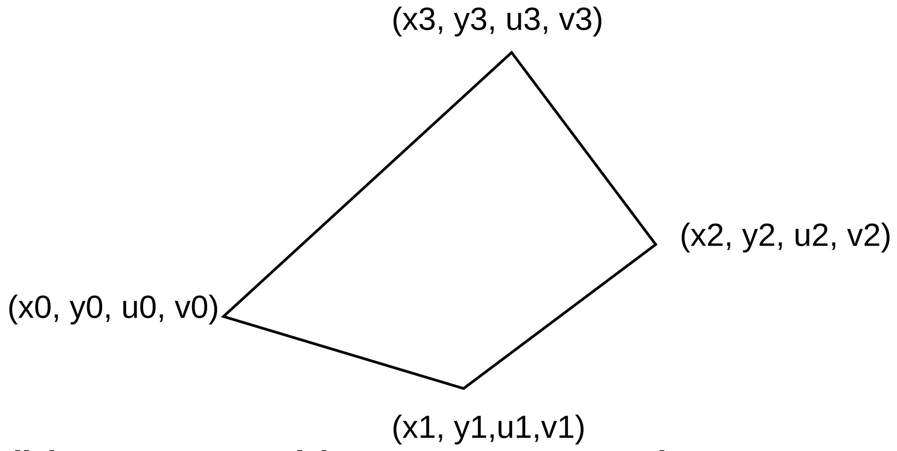
If $\sum x = 0$ *and* $\sum y = 0$, the (x,y) quadrilateral is a parallelogram, and the matrix reduces to an affine matrix:

$$\mathbf{R}_{affine} = \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & 0 \\ x_3 - x_0 & y_3 - y_0 & 0 \\ x_0 & y_0 & 1 \end{bmatrix}$$

These 3x3 matrices represent a projective mapping from texture to screen space as:

$$[x \quad y \quad q] = [u \quad v \quad 1]\mathbf{R}$$

We can do a similar sort of mapping from a unit square to an arbitrary quadrilateral in (u,v).



If we call this mapping $\mathbf{R_u}$ and the previous mapping $\mathbf{R_x}$, then we can compute a combined mapping from (x,y) to unit square to (u,v) as:

$$[x \quad y \quad q] = [s \quad t \quad 1]\mathbf{R}_x$$

$$[u \quad v \quad w] = [s \quad t \quad 1]\mathbf{R}_u$$

$$[u \quad v \quad w] = [x \quad y \quad 1]\mathbf{R}_x^{adj}\mathbf{R}_u = [x \quad y \quad 1]\mathbf{R}$$

or

$$\mathbf{R} = \mathbf{R}_x^{adj}\mathbf{R}_u.$$

10.2. 2D-3D Projective Mapping from World Space Correspondence

We can compute the projective mapping matrix from 3 points **R** by Gaussian elimination:

$$\begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix} = \begin{bmatrix} u_0 & v_0 & 1 \\ u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}$$

We will later embed this into a 3x4 matrix as follows:

$$\begin{bmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \end{bmatrix} = \begin{bmatrix} u_0 & v_0 & 1 \\ u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & 0 \\ r_{10} & r_{11} & r_{12} & 0 \\ r_{20} & r_{21} & r_{22} & 1 \end{bmatrix}$$

With 4 points, we compute a projective correspondence between (x,y,z) and a unit square. We then concatenate that with a mapping from the unit square to the given (u,v)'s as in the previous section.

The projective mapping from four coplanar points in (x,y,z) to the unit square is given by:

$$\begin{bmatrix} x_1 - x_0 + gx_1 & y_1 - y_0 + gy_1 & z_1 - z_0 + gz_1 & g \\ x_3 - x_0 + hx_3 & y_3 - y_0 + hy_3 & z_3 - z_0 + hz_3 & h \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

where

$$g = \frac{d - s_n}{r_n + s_n - d}, \qquad h = \frac{d - r_n}{r_n + s_n - d}$$

$$r_n = \begin{vmatrix} d_{12} & d_{13} \\ d_{32} & d_{33} \end{vmatrix}, \quad s_n = \begin{vmatrix} d_{11} & d_{12} \\ d_{13} & d_{32} \end{vmatrix}, \quad d = \begin{vmatrix} d_{11} & d_{13} \\ d_{13} & d_{33} \end{vmatrix}$$

$$d_{11} = \mathbf{v}_1 \cdot \mathbf{v}_1, \quad d_{12} = \mathbf{v}_1 \cdot \mathbf{v}_2, \quad d_{13} = \mathbf{v}_1 \cdot \mathbf{v}_3, \quad d_{32} = \mathbf{v}_3 \cdot \mathbf{v}_2$$

$$\mathbf{v}_1 = \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \end{bmatrix}$$

$$\mathbf{v}_2 = \begin{bmatrix} x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \end{bmatrix}$$

$$\mathbf{v}_3 = \begin{bmatrix} x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \end{bmatrix}$$

We can also generalize our 2D results to 3D, yielding an alternate expression for *g* and *h*:

$$g = \frac{(\Delta_{01} \times \Delta_{32}) \cdot (\Delta_{01} \times \Delta_{03})}{(\Delta_{12} \times \Delta_{32}) \cdot (\Delta_{01} \times \Delta_{03})}, \qquad h = \frac{(\Delta_{12} \times \Delta_{03}) \cdot (\Delta_{01} \times \Delta_{03})}{(\Delta_{12} \times \Delta_{32}) \cdot (\Delta_{01} \times \Delta_{03})}$$

where we have chosen $\Delta_{01} \times \Delta_{03}$ as the quadrilateral normal, to give identical results as the previous expression. Another equally valid expression uses $\Delta_{12} \times \Delta_{32}$ as the quadrilateral normal, and results in the slightly simpler expression:

$$g = \frac{(\Delta_{01} \times \Delta_{32}) \cdot (\Delta_{12} \times \Delta_{32})}{|\Delta_{12} \times \Delta_{32}|^2}, \qquad h = \frac{(\Delta_{12} \times \Delta_{03}) \cdot (\Delta_{12} \times \Delta_{32})}{|\Delta_{12} \times \Delta_{32}|^2}$$

In the graphics pipeline, homogeneous vectors are transformed by modeling (**M**), viewing (**V**), clipping (**C**), and screen-mapping (**S**) matrices, whose concatenation is represented by the matrix **A**. The mapping from texture to screen coordinates is then:

$$\mathbf{x}' = \mathbf{x}\underbrace{\mathbf{MVCS}}_{\mathbf{A}} = \mathbf{uR}\underbrace{\mathbf{MVCS}}_{\mathbf{A}}$$

If we then toss out the resultant third component with the matrix P:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

we have

$$\mathbf{x}'' = \mathbf{x}\underbrace{\mathbf{MVCS}}_{\mathbf{A}}\mathbf{P} = \mathbf{u}\underbrace{\mathbf{RMVCSP}}_{\mathbf{B}} = \mathbf{uB}$$

If we then toss out the composite transformation is:

$$\mathbf{x}'' = [i \quad j \quad h] = \underbrace{\mathbf{u}}_{1x3}\underbrace{\mathbf{R}}_{3x4}\underbrace{\mathbf{A}}_{4x4}\underbrace{\mathbf{P}}_{4x3} = [u \quad v \quad w]\mathbf{B}$$

where

$$\mathbf{B} = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

This is a projective mapping that takes parametric space to screen space. Its inverse is more useful for texture mapping, because it takes screen space to parametric space. However, since B specifies a projective mapping, its adjoint can be used instead of the inverse [PENNA]. In fact, the adjoint is preferable, because it always exists, whereas the inverse does not always exist. The computation of the adjoint is relatively simple:

$$\mathbf{B}^{adj} = \begin{bmatrix} \tilde{u}_x & \tilde{v}_x & \tilde{w}_x \\ \tilde{u}_y & \tilde{v}_y & \tilde{w}_y \\ \tilde{u}_h & \tilde{v}_h & \tilde{w}_h \end{bmatrix} = \begin{bmatrix} b_{22}b_{11} - b_{12}b_{21} & b_{02}b_{21} - b_{22}b_{01} & b_{12}b_{01} - b_{02}b_{11} \\ b_{12}b_{20} - b_{10}b_{22} & b_{22}b_{00} - b_{02}b_{20} & b_{10}b_{02} - b_{12}b_{00} \\ b_{10}b_{21} - b_{20}b_{11} & b_{20}b_{01} - b_{00}b_{21} & b_{00}b_{11} - b_{10}b_{01} \end{bmatrix}$$

Parametric coordinates may then be obtained from screen coordinates from first calculating:

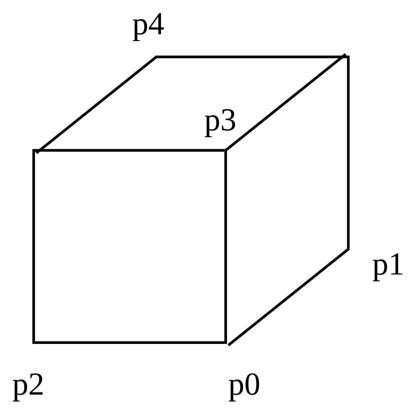
$$[\tilde{u} \quad \tilde{v} \quad \tilde{w}] = [i \quad j \quad 1]\mathbf{B}^{adj}$$

and then dividing by w:

$$u = \frac{\tilde{u}}{\tilde{w}}, \quad v = \frac{\tilde{v}}{\tilde{w}}$$

10.3. 3D-3D Projective Mappings

We desire to find the 4x4 projective transformation that maps the unit cube to a frustum, as found in a camera specification.



Assuming that it has the form:

$$\mathbf{R} = \begin{bmatrix} x_1 - x_0 + fx_1 & y_1 - y_0 + fy_1 & z_1 - z_0 + fz_1 & f \\ x_2 - x_0 + gx_2 & y_2 - y_0 + gy_2 & z_2 - z_0 + gz_2 & g \\ x_3 - x_0 + hx_3 & y_3 - y_0 + gy_3 & z_3 - z_0 + hz_3 & h \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

we find that *f*, *g*, and *h* are given by solutions to the equation:

$$\begin{bmatrix} x_4 - x_1 & x_4 - x_2 & x_4 - x_3 \\ y_4 - y_1 & y_4 - y_2 & y_4 - y_3 \\ z_4 - z_1 & z_4 - z_2 & z_4 - z_3 \end{bmatrix} \begin{bmatrix} f \\ g \\ h \end{bmatrix} = \begin{bmatrix} x_1 + x_2 + x_3 - x_4 - 2x_0 \\ y_1 + y_2 + y_3 - y_4 - 2y_0 \\ z_1 + z_2 + z_3 - z_4 - 2z_0 \end{bmatrix}$$

10.4. Determining the Jacobian Matrix of a Projective Map

In order to find the jacobian matrix for a projective map, we invoke the quotient rule for derivatives.

11. Functional Parametrization

We now address the issue of applying a parametrization (or reparametrization) to a surface by means of a function. In the purely general case, we have

$$(u,v) = f(\mathbf{N})$$

where **N** is the neighborhood of a point, and contains:

• position
• normal
• tangents
• [other] parametrizations, etc.

When the parametrization is a function only of the normal, then we have [a variant of] reflection-mapping.
First, we consider when the parametrization is a function only of the position.

11.1. Parametrization as a Function of Position

Here, we have:

$$(u,v) = f(x,y,z)$$

Some obvious candidates are projection of the parametrization of a circumscribing cylinder or sphere onto the object. The problem we address here is how to determine the tangent vectors. We will assume that we know the normal at each point.

We can compute the jacobian

$$\frac{\partial(u,v)}{\partial(x,y,z)} = \begin{bmatrix} u_x & v_x \\ u_y & v_y \\ u_z & v_z \end{bmatrix}$$

but what we want is sort of the inverse of that, namely:

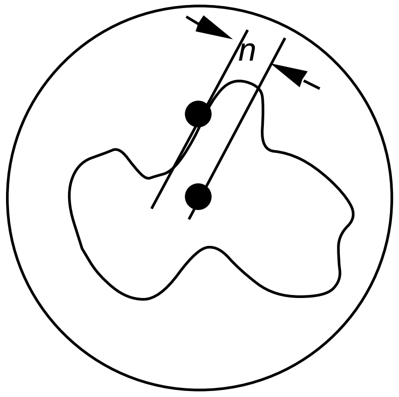
$$\frac{\partial(x,y,z)}{\partial(u,v)} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{bmatrix}$$

We introduce another component into the function so that the jacobian is square and can be inverted. If we choose a component that it is collinear with the normal, then if we invert the jacobian, the resultant u and v derivatives are the desired tangents.

For cylindrical and spherical projections, one such suitable value is the normal distance from the center of projection:

$$(u,v,n) = f(x,y,z)$$

This is illustrated in the diagram below for a cylindrical projection:



We choose a class of functions that have the normalized normal as its gradient.

From the jacobian:

$$\frac{\partial(u,v,n)}{\partial(x,y,z)} = \begin{bmatrix} u_x & v_x & n_x \\ u_y & v_y & n_y \\ u_z & v_z & n_z \end{bmatrix}$$

we compute its inverse:

$$\frac{\partial(x,y,z)}{\partial(u,v,n)} = \begin{bmatrix} x_u & y_u & z_u \\ \hline x_v & y_v & z_v \\ \hline x_n & y_n & z_n \end{bmatrix} = \left[\begin{array}{c|c|c} u_x & v_x & n_x \\ u_y & v_y & n_y \\ u_z & v_z & n_z \end{array}\right]^{-1}$$

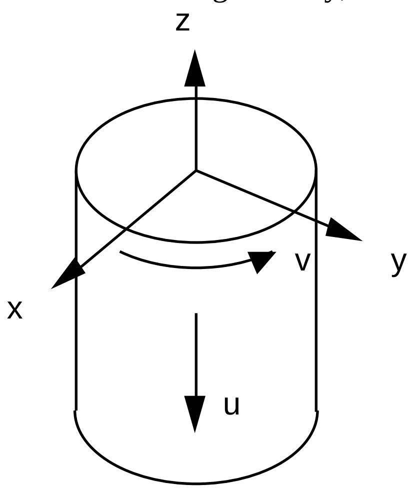
Note that the top row is the derivative of (*x*,*y*,*z*) with respect to *u*, holding *v* and *n* constant. The insight into this is that when *n* is constant (and equal to zero), we are in the tangent plane.

This allows us to now toss away the derivatives of *n*, yielding the tangents w.r.t. *u* and *v*:

$$\frac{\partial(x,y,z)}{\partial(u,v)} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \end{bmatrix}$$

11.2. Cylindrical Mapping

Without loss of generality, we assume that the origin is the center of projection.



Here, the projection equations are given by:

$$u = 1 - z$$

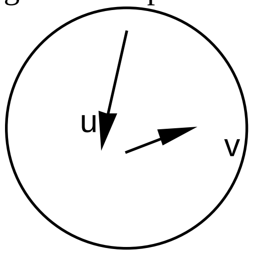
$$v = \mathrm{atan2}(y,x) = \tan^{-1}\left(\frac{y}{x}\right)$$

which has the jacobian matrix[2]:

$$\frac{\partial(u,v)}{\partial(x,y,z)} = \begin{bmatrix} 0 & -\frac{y}{x^2+y^2} \\ 0 & \frac{x}{x^2+y^2} \\ -1 & 0 \end{bmatrix}$$

11.3. Spherical Mapping

The origin of the sphere is the center of projection:



[2]This was computed in Mathematica™ with **Outer[D, {1-z, ArcTan[y/x]}, {x, y, z}]** and transposed.

$$u = -\operatorname{atan2}\left(z, \sqrt{x^2+y^2}\right) = \tan^{-1}\left(\frac{z}{\sqrt{x^2+y^2}}\right)$$

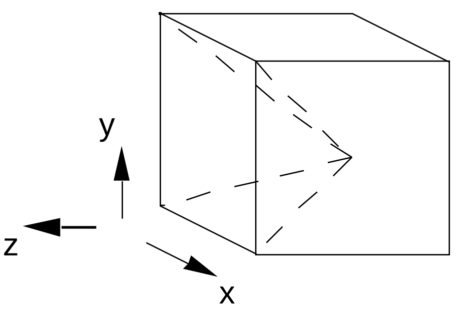
$$v = \operatorname{atan2}(y,x) = \tan^{-1}\left(\frac{y}{x}\right)$$

whose jacobian matrix is[3]:

$$\frac{\partial(u,v)}{\partial(x,y,z)} = \begin{bmatrix} -\frac{xz}{\sqrt{x^2+y^2}\left(x^2+y^2+z^2\right)} & -\frac{y}{x^2+y^2} \\ -\frac{yz}{\sqrt{x^2+y^2}\left(x^2+y^2+z^2\right)} & \frac{x}{x^2+y^2} \\ \frac{\sqrt{x^2+y^2}}{\left(x^2+y^2+z^2\right)} & 0 \end{bmatrix}$$

11.4. Cube Mapping

Here, we project the surface point onto one of the eight cube faces. In other words, the first part of the parametrization is to determine which of the 8 pyramids a particular point belongs to.



In the above diagram, suppose we use the function

$$u = \frac{x}{2z} + \frac{1}{2}, \qquad v = \frac{y}{2z} + \frac{1}{2}$$

for the left face. This gives the jacobian[4]:

$$\frac{\partial(u,v)}{\partial(x,y,z)} = \begin{bmatrix} \frac{1}{2z} & 0 \\ 0 & \frac{1}{2z} \\ -\frac{x}{2z^2} & -\frac{y}{2z^2} \end{bmatrix}$$

The other faces would have different but similar parametrizations and jacobians.

[3]Likewise, Mathematica gives this with **Outer[D, {ArcTan[z/Sqrt[x^2+y^2]], ArcTan[y/x]}, {x, y, z}]** and transposition.

[4]Mathematica: Outer[D, {(x/(2 z))+(1/2), (y/(2 z))+(1/2)}, {x, y, z}] transposed.

11.5. Planar Mapping

This is a planar projection, such as:

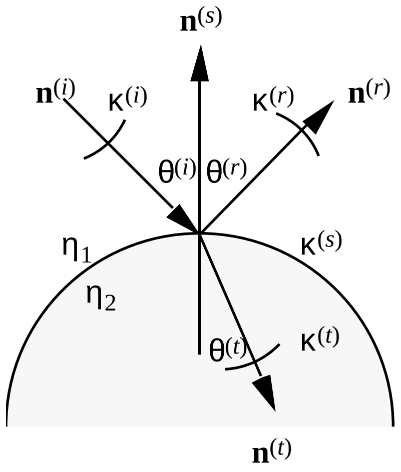
$$u = \frac{x}{2} + \frac{1}{2}, \qquad v = \frac{y}{2} + \frac{1}{2}$$

which has the jacobian:

$$\frac{\partial(u,v)}{\partial(x,y,z)} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \\ 0 & 0 \end{bmatrix}$$

12. Wavefront Tracing

[Mitchell and Hanrahan] and [Glassner] give the equations for the propagation of a ray as it interacts with a surface.



The reflected ray is given by:

$$\begin{aligned} \mathbf{n}^{(r)} &= \mathbf{n}^{(i)} + 2\cos\theta_i \mathbf{n}^{(s)} \\ &= \mathbf{n}^{(i)} - 2\left(\mathbf{n}^{(i)} \cdot \mathbf{n}^{(s)}\right)\mathbf{n}^{(s)} \end{aligned}$$

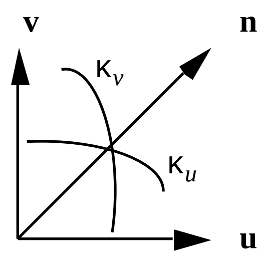
and the transmitted (refracted) ray is given by:

$$\mathbf{n}^{(t)} = \eta \mathbf{n}^{(i)} + \gamma \mathbf{n}^{(s)}$$

where

$$\begin{aligned} \gamma &= \eta \cos\theta_i - \sqrt{1 + \eta^2\left[\cos^2\theta_i - 1\right]} \\ &= \eta\left(\mathbf{n}^{(i)} \cdot \mathbf{n}^{(s)}\right) + \sqrt{1 + \eta^2\left[\left(\mathbf{n}^{(i)} \cdot \mathbf{n}^{(s)}\right)^2 - 1\right]} \end{aligned}$$

[Mitchell and Hanrahan] extend the results for rays to propagating wavefronts. A wavefront is represented by an orthonormal frame and two principal curvatures:



The wavefront propagates in a direction **n**, and has two principal directions **u** and **v**, each with a curvature κ_u and κ_v.
For an orthographic camera, the curvature of the wavefront is zero in all directions at all distances:

$$\kappa = 0$$

For a projective camera, the curvature in all directions is constant at a given distance:

$$\kappa = -\frac{1}{d}$$

Wavefront propagation by transfer through free space is given by:

$$\kappa_u' = \frac{\kappa_u}{1 - d\kappa_u}$$

$$\kappa_v' = \frac{\kappa_v}{1 - d\kappa_v}$$

where *d* is the propagation distance of the wavefront.

Reflection and refraction are more complicated. First we compute a new reference frame, and compute the *wavefront* curvatures in this new frame. The direction

$$\mathbf{u}' = \frac{\mathbf{n}^{(i)} \times \mathbf{n}^{(s)}}{\left|\mathbf{n}^{(i)} \times \mathbf{n}^{(s)}\right|}$$

is tangent to both the incident wavefront and the surface. Of course, the denominator goes to zero if the incident wavefront and the surface normal are collinear, so in this case, any vector in the common tangent plane may be used as the **u**´ vector. Although not necessary at this point, the vector **v**´ is given by:

$$\mathbf{v}' = (\mathbf{u} \times \mathbf{u}') \times \mathbf{v} + (\mathbf{u} \cdot \mathbf{u}')\mathbf{v} = (\mathbf{u} \cdot \mathbf{u}')\mathbf{v} - (\mathbf{v} \bullet \mathbf{u}')\mathbf{u}$$

We can find the curvatures of the wavefront in this direction with Euler's formula:

$$\kappa_{u'} = \kappa_u \cos^2\theta + \kappa_v \sin^2\theta$$
$$\kappa_{v'} = \kappa_u \sin^2\theta + \kappa_v \cos^2\theta$$
$$\kappa_{u'v'} = (\kappa_u - \kappa_v)\cos\theta\sin\theta$$

where the trigonometric terms can be computed with:

$$\cos\theta = \mathbf{u}' \bullet \mathbf{u}$$
$$\sin\theta = \mathbf{u}' \bullet \mathbf{v}$$

We next need to determine the curvatures of the *surface* in this new frame. For a parametric surface, we have the parametric tangents $\mathbf{x}_u$ and $\mathbf{x}_v$. We can determine the coordinates of the new frame in the surface's parametric coordinates with:

$$\mathbf{u}'' = \begin{bmatrix}\mathbf{u}' \bullet \mathbf{x}_u & \mathbf{u}' \bullet \mathbf{x}_v\end{bmatrix}\begin{bmatrix}\mathbf{x}_u \bullet \mathbf{x}_u & \mathbf{x}_u \bullet \mathbf{x}_v \\ \mathbf{x}_v \bullet \mathbf{x}_u & \mathbf{x}_v \bullet \mathbf{x}_v\end{bmatrix}^{-1}$$
$$= \mathbf{u}'\begin{bmatrix}\mathbf{x}_u{}^T & \mathbf{x}_v{}^T\end{bmatrix}\begin{bmatrix}\mathbf{x}_u \bullet \mathbf{x}_u & \mathbf{x}_u \bullet \mathbf{x}_v \\ \mathbf{x}_v \bullet \mathbf{x}_u & \mathbf{x}_v \bullet \mathbf{x}_v\end{bmatrix}^{-1}$$

where **u**´´ are the 2-coordinates of the **u**´ vector in the surface's parametric coordinates. The inverted matrix on the right is necessary because the surface's parametric coordinates are not orthonormal. An orthogonal vector to **u**´ in the surface tangent plane is

The curvature in the **u**´´ direction is given by:

$$\kappa_{\mathbf{u}''} = \mathbf{u}''\mathbf{D}\,\mathbf{u}''^T$$

where **D** is the curvature tensor of the parametric surface. The metric tensor is not needed because the **u**´´ vector is a unit vector, as measured in world space.

We need to compute the κ_v and κ_{uv} curvatures as well, in this new frame. A vector orthogonal to the **u**´ vector in the surface tangent plane is

$$\mathbf{u}' \times \mathbf{n}^{(s)}$$

so that we can transform the curvature tensor as:

$$\mathbf{D}' = \mathbf{U}\mathbf{D}\mathbf{U}^T$$

where

$$\mathbf{U} = \begin{bmatrix}\mathbf{u}' \\ \mathbf{u}' \times \mathbf{n}^{(s)}\end{bmatrix}\begin{bmatrix}\mathbf{x}_u{}^T & \mathbf{x}_v{}^T\end{bmatrix}\begin{bmatrix}\mathbf{x}_u \bullet \mathbf{x}_u & \mathbf{x}_u \bullet \mathbf{x}_v \\ \mathbf{x}_v \bullet \mathbf{x}_u & \mathbf{x}_v \bullet \mathbf{x}_v\end{bmatrix}^{-1}$$

We now have enough information to determine the curvatures in the new frame. The curvatures for reflection are:

$$\kappa_u^{(r)} = \kappa_u^{(i)} + 2\cos\theta_i \kappa_u^{(s)}$$
$$\kappa_{uv}^{(r)} = -\kappa_{uv}^{(i)} - 2\kappa_{uv}^{(s)}$$
$$\kappa_v^{(r)} = \kappa_v^{(i)} + \left(2/\cos\theta_i\right)\kappa_v^{(s)}$$

and those for transmission (refraction) are (Mitchell-Hanrahan have an error here):

$$\kappa_u^{(t)} = \eta\kappa_u^{(i)} + \gamma\kappa_u^{(s)}$$
$$\kappa_{uv}^{(t)} = \eta\frac{\cos\theta_i}{\cos\theta_t}\kappa_{uv}^{(i)} + \frac{\gamma}{\cos\theta_t}\kappa_{uv}^{(s)}$$
$$\kappa_v^{(t)} = \eta\frac{\cos^2\theta_i}{\cos^2\theta_t}\kappa_v^{(i)} + \frac{\gamma}{\cos^2\theta_t}\kappa_v^{(s)}$$

where the cosines are given by:

$$\cos\theta_i = -\mathbf{n}^{(i)} \cdot \mathbf{n}^{(s)}$$
$$\cos\theta_t = -\mathbf{n}^{(t)} \cdot \mathbf{n}^{(s)}$$

All that is left is to find the principal axes and principal curvatures.
Eigenvalue analysis yields:

$$\kappa_1 = \frac{\kappa_u + \kappa_v + \sqrt{4\kappa_{uv}^2 + \left(\kappa_u - \kappa_v\right)^2}}{2}$$
$$\kappa_2 = \frac{\kappa_u + \kappa_v - \sqrt{4\kappa_{uv}^2 + \left(\kappa_u - \kappa_v\right)^2}}{2}$$
$$\mathbf{w}_1 = \begin{bmatrix} \dfrac{-2\kappa_{uv}}{\kappa_u - \kappa_v - \sqrt{4\kappa_{uv}^2 + \left(\kappa_u - \kappa_v\right)^2}} & 1 \end{bmatrix}$$
$$\mathbf{w}_2 = \begin{bmatrix} \dfrac{-2\kappa_{uv}}{\kappa_u - \kappa_v + \sqrt{4\kappa_{uv}^2 + \left(\kappa_u - \kappa_v\right)^2}} & 1 \end{bmatrix}$$

if κ_{uv} is zero, then we already have the principal axes and curvatures, otherwise a rotation of up to 45° is performed. The new axes are given by $\mathbf{w}_1$ and $\mathbf{w}_2$.

12.1. Anti-Aliasing Specular Reflections and Refractions using Wavefront Tracing

We want to trace the shape of a pixel throughout the system as it reflects from and refracts through surfaces. In short, the size of a pixel varies inversely with the curvature. At the last surface, the pixel is projected onto the surface and appropriate anti-aliasing is done.
(more detail…).
First, we need to evaluate the ray wavefront on the surface. Recall that it is given by:

$$\begin{bmatrix} x & y & z \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix} + t\left(\begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix} - b_{23}\begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix}\right)$$

At the point (x_0, y_0, z_0), (i.e. t=0), the derivatives w.r.t. *i* and *j* are

$$\mathbf{J} = \left\{ \begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix} \begin{bmatrix} a_{03} \\ a_{13} \\ a_{23} \\ a_{33} \end{bmatrix} \right\} \left\{ \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{bmatrix} - \begin{bmatrix} b_{03} \\ b_{13} \end{bmatrix} \begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix} \right\}$$

The normal of the ray is given by:

$$\hat{\mathbf{n}} = \frac{\mathbf{n}}{|\mathbf{n}|}$$

where

$$\mathbf{n} = \begin{bmatrix} b_{20} & b_{21} & b_{22} \end{bmatrix} - b_{23}\begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix}$$

and the curvatures in the *i* and *j* directions are given by:

$$\kappa_i = \frac{b_{23}}{|\mathbf{n}|}\left\{1 - \frac{(\mathbf{J}_i \bullet \hat{\mathbf{n}})^2}{\mathbf{J}_i \bullet \mathbf{J}_i}\right\} \qquad \kappa_j = \frac{b_{23}}{|\mathbf{n}|}\left\{1 - \frac{(\mathbf{J}_j \bullet \hat{\mathbf{n}})^2}{\mathbf{J}_j \bullet \mathbf{J}_j}\right\}$$

13. Appendix: Norms for Jacobian Matrices

Here is a study of several norms for jacobian matrices. Note that these formulations are for jacobian matrices that are expected to be multiplied by row vectors, not column vectors as found in textbooks.

$$\left\| [a_{ij}] \right\|_1 = \max_i \sum_j |a_{ij}|$$

$$\left\| [a_{ij}] \right\|_2 = \max_i |\lambda_i|; \quad \left\| [a_{ij}]_{2x2} \right\|_2 = \left| \frac{|a_{00} + a_{11}| + \sqrt{(a_{00} + a_{11})^2 - 4(a_{00}a_{11} - a_{01}a_{10})}}{2} \right|$$

$$\left\| [a_{ij}] \right\|_{Heckbert} = \max_i \sqrt{\sum_j a_{ij}^2}$$

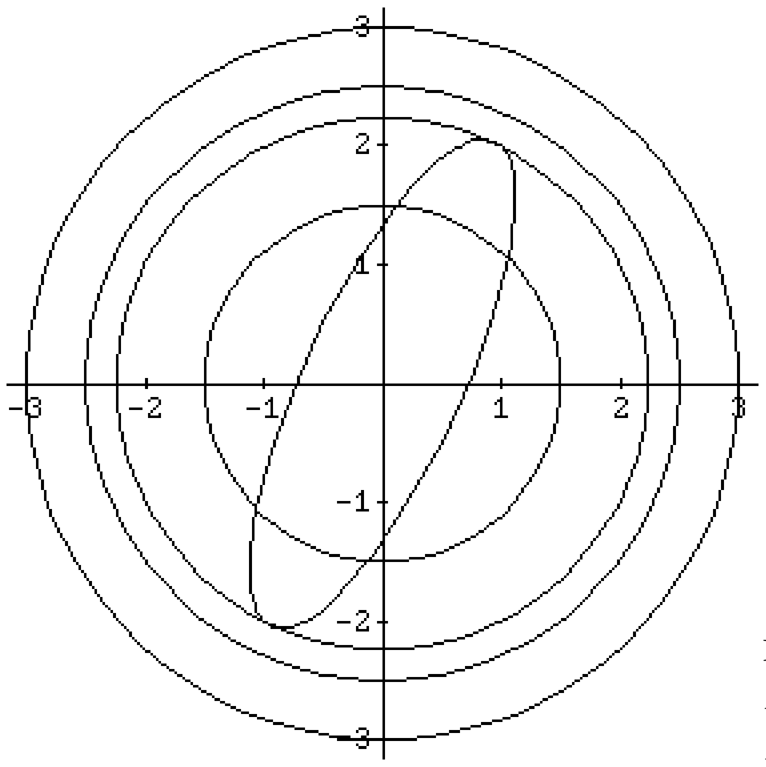
$$\left\| [a_{ij}] \right\|_\infty = \max_j \sum_i |a_{ij}|$$

The L1 norm is the maximum of the sums of the absolute value of the rows (max L1 norm of rows).

The L∞ norm is the maximum of the sums of the absolute value of the columns. (max L1 norm of columns)

The L2 norm is the maximum eigenvalue of the matrix.

The LHeckbert norm is the maximum of the Euclidean norms of the rows (max L2 norm of the rows).



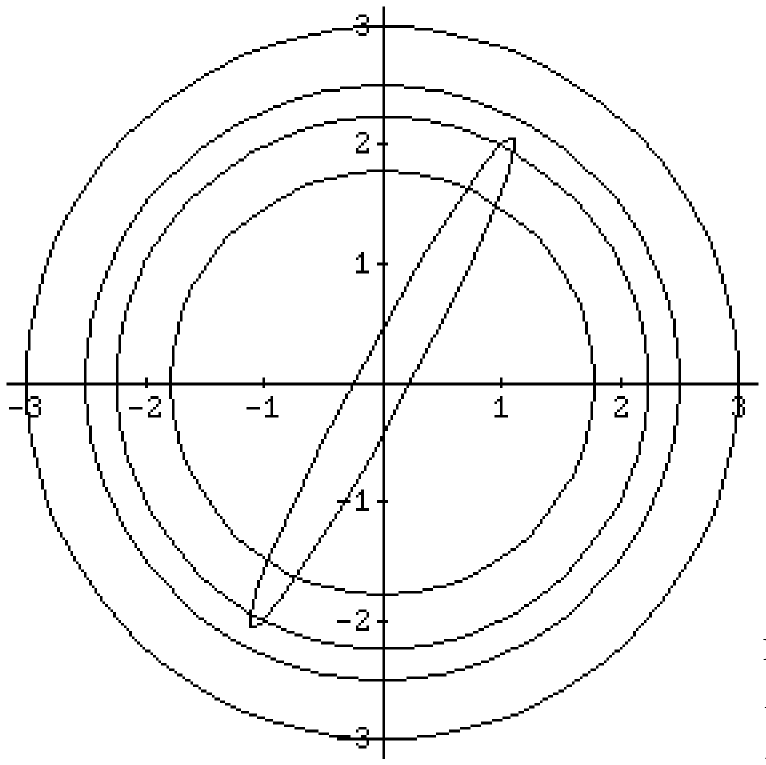
From outer to inner:

L_1

L_∞

$L_{Heckbert}$

L_2



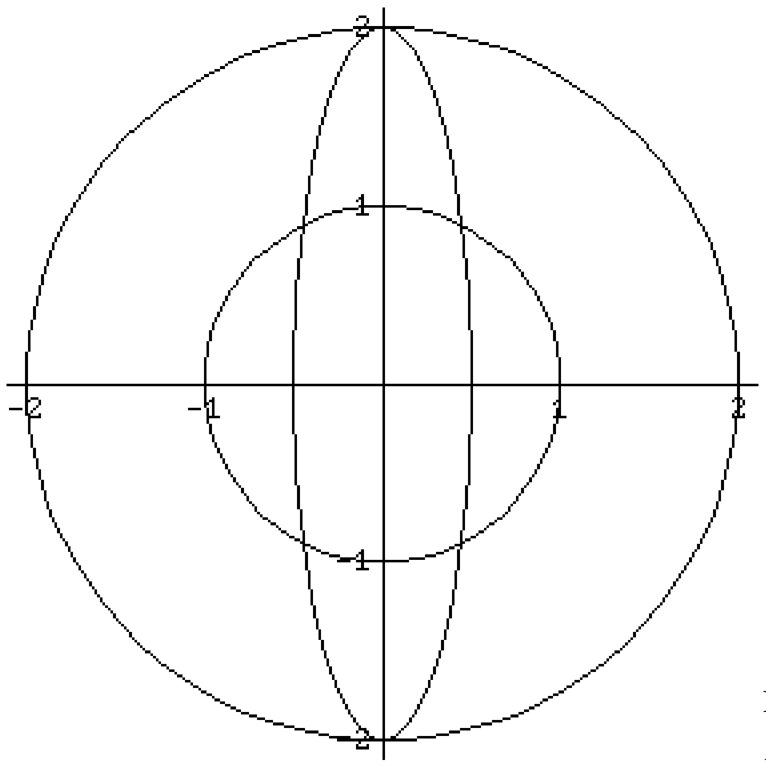
From outer to inner:

L_1

L_∞

$L_{Heckbert}$

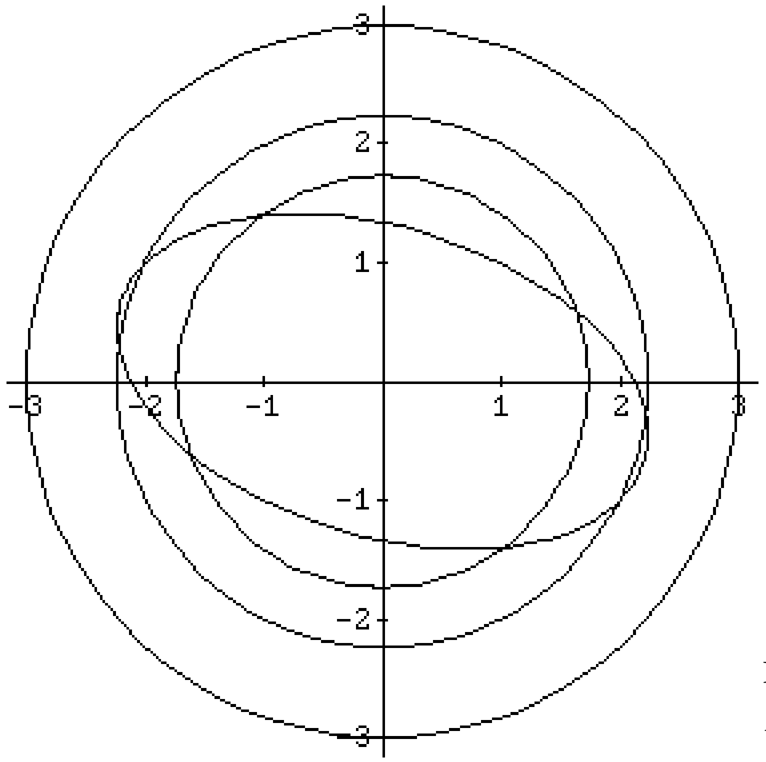
L_2



From outer to inner:

L_1, L_∞, $L_{Heckbert}$

L_2



From outer to inner:

L_1, L_∞

$L_{Heckbert}$

L_2

From these pictures, we see that the L_1 norm is the most conservative, in that it always yields the biggest circle. The L_∞ norm is not far from it. Both of them always circumscribe the ellipse. They are both quite anisotropic in that the resultant diameter is larger for an inclined ellipse than an upright (or flat) ellipse.

The $L_{Heckbert}$ norm solves the anisotropy problem, and results in a diameter that is very close to the length of the major axis.

The L_2 norm is also relatively isotropic, and it strives to achieve a balance between aliasing and blurriness. The area inside the ellipse and outside the circle represents aliasing; the area inside the circle and outside the ellipse represents blurriness.