5. Appendices

The appendices of this paper provide the Cg fragment shader code and the math to convert texture coordinates in continuous cube mapping. The math for this process was first presented in [Grimm 04].

Appendix A gives the Cg fragment shader code. This takes a reflected vector and uses the *uv* coordinate equations described in Section 3 to perform texture lookups. The *uv* coordinate equations from Section 3 are condensed versions of the math presented in Appendx B. The math in Appendix B is reversible. This process reversed is provided in Appendix C.

A. Cg Fragment Shader

```
float4 Rotate_R(float3 R)
{
    // makes the program think we're only computing the
    // positive-y texture
    float x, y, z, texIndex;
    float L = sqrt(2.0) / 2.0;

    // create the normals to check against
    float3 N1 = float3(L, L, 0.0);
    float3 N2 = float3(-L, L, 0.0);
    float3 N3 = float3(0.0, L, L);
    float3 N4 = float3(0.0, L, -L);
    float3 N5 = float3(L, 0.0, L);
    float3 N6 = float3(-L, 0.0, L);

    R = normalize(R);

    bool check1 = ceil(dot(N1, R));
    bool check2 = ceil(dot(N2, R));
    bool check3 = ceil(dot(N3, R));
    bool check4 = ceil(dot(N4, R));
    bool check5 = ceil(dot(N5, R));
    bool check6 = ceil(dot(N6, R));

    // positive y
    if (check1 && check2 && check3 && check4)
    {
        // do nothing
        x = R.x;
```

```
    y = R.y;
    z = R.z;
    texIndex = 0.0;
}
// negative y
else if (!check1 && !check2 && !check3 && !check4)
{
    // rotate around x, 180 degrees
    x = R.x;
    y = -1.0*R.y;
    z = -1.0*R.z;
    texIndex = 1.0;
}
// positive z
else if (check5 && check6 && check3 && !check4)
{
    // rotate around x, 90 degrees
    x = R.x;
    y = R.z;
    z = -1.0*R.y;
    texIndex = 2.0;
}
// negative z
else if (!check5 && !check6 && !check3 && check4)
{
    // rotate around x, -90 degrees
    x = R.x;
    y = -1.0*R.z;
    z = R.y;
    texIndex = 3.0;
}
// negative x
else if (!check5 && check6 && !check1 && check2)
{
    // rotate around z, -90 degrees
    x = R.y;
    y = -1.0*R.x;
    z = R.z;
    texIndex = 4.0;
}
// positive x
else if (check5 && !check6 && check1 && !check2)
{
```

```
        // rotate around z, 90 degrees
        x = -1.0*R.y;
        y = R.x;
        z = R.z;
        texIndex = 5.0;
    }
    else
    {
        // do nothing
        x = 0.0;
        y = 0.0;
        z = 0.0;
        texIndex = 6.0;
    }

    return float4(x, y, z, texIndex);
}

void frag_program(  float3 R           : TEXCOORD1,

                out float4 oColor      : COLOR,

            uniform sampler2D decalX,
            uniform sampler2D decalNegX,
            uniform sampler2D decalY,
            uniform sampler2D decalNegY,
            uniform sampler2D decalZ,
            uniform sampler2D decalNegZ )
{
    float4 temp = Rotate_R(R);
    float3 rotatedR = temp;
    float  texIndex = temp[3];

    float x = rotatedR.x;
    float y = rotatedR.y;
    float z = rotatedR.z;

    float S1 = atan2((x / y), sqrt(2.0));
    float S2 = 2.0 * asin(1.0 / sqrt(3.0));

    float S = 0.5 - S1/S2;
    float T = ( 1.0 + (asin(z) /
       ( asin(1.0 / (sqrt (2.0 + (x/y)*(x/y) ) ) ) ) ) ) / 2.0;
```

```
    // Fetch relected environment color
    if (texIndex == 0.0)
    {
       oColor = tex2D(decalY, float2(S, T));
    }
    else if (texIndex == 1.0)
    {
       oColor = tex2D(decalNegY, float2(S, T));
    }
    else if (texIndex == 2.0)
    {
       oColor = tex2D(decalZ, float2(S, T));
    }
    else if (texIndex == 3.0)
    {
       oColor = tex2D(decalNegZ, float2(S, T));
    }
    else if (texIndex == 4.0)
    {
       oColor = tex2D(decalNegX, float2(S, T));
    }
    else if (texIndex == 5.0)
    {
       oColor = tex2D(decalX, float2(S, T));
    }
    else if (texIndex == 6.0)
    {
       oColor = float4(0.0, 0.0, 0.0, 1.0);
    }
}
```

B. Math to convert from Rx,Ry,Rz to square texture coordinates

```
dPi = 3.0 * PI / 4.0

switch(face}
case 0:
    dS = atan2(Ry, Rx)
    dT = asin(Rz)
    dS = dS / PI
    dT = (dT + dPi / 2.0) / dPi
```

```
    break
case 1:
    dS = atan2(Ry, Rx)
    dT = asin(Rz)
    dS = 1.0 + dS / PI
    dT = (dT + dPi / 2.0) / dPi
    break
case 2:
    dS = atan2(Rx, Rz)
    dT = asin(Ry)
    dS = dS / PI
    dT = (dT + dPi / 2.0) / dPi
    break
case 3:
    dS = atan2(Rx, Rz)
    dT = asin(Ry)
    dS = 1.0 + dS / PI
    dT = (dT + dPi / 2.0) / dPi
    break
case 4:
    dS = atan2(Rz, Ry)
    dT = asin(Rx)
    dS = dS / PI
    dT = (dT + dPi / 2.0) / dPi
    break
case 5:
    dS = atan2(Rz, Ry)
    dT = asin(Rx)
    dS = 1.0 + dS / PI
    dT = (dT + dPi / 2.0) / dPi
    break

if (dS < -0.5)
    dS = dS + 2.0
else if (dS > 1.5)
    dS = dS - 2.0


c_dv1 = asin( -sqrt(1/3) ) * 4 / (3 * PI) + 0.5
c_dv2 = asin( sqrt(1/3) ) * 4 / (3 * PI) + 0.5


dSP = (0.75 + 1.0) * PI
```

```
dYdX = tan(PI * dS)
dA = dYdX / cos(dSP)
dTP = atan(1 / dA)
dPerc = (dTP + dPi / 2) / dPi

u = (dPerc - c_dv2) / (c_dv1 - c_dv2)

dZ = sin(dSP) * cos(dTP)
dAng2 = asain(dZ)
dTMe = (dAng2 + dPi / 2) / dPi

v = (dT - dTMe) / (1 - 2 * dTMe)
```

C. Math to convert from square texture coordinates to Rx, Ry, Rz

```
dPi = 3.0 * PI / 4.0

c_dv1 = asin( -sqrt(1/3) ) * 4 / (3 * PI) + 0.5
c_dv2 = asin( sqrt(1/3) ) * 4 / (3 * PI) + 0.5

dPercV = c_dv2 * (1.0 - u) + c_dv1 * u

dSP = (0.75 + 1.0) * PI
dTP = (dPercV * dPi - dPi / 2)

dY = cos(dSP) * cos(dTP)
dX = sin(dTP)
dZ = sin(dSP) * cos(dTP)
dAng1 = atan2(dY, dX)
dAng2 = asin(dZ)
dSMe = dAng1 / PI
dTMe = (dAng2 + dPi / 2) / dPi


switch(face}
case 0:
    dS = dU * PI
    dT = dV * dPi - dPi / 2

    Rx = cos(dS) * dCT
```

```
    Ry = sin(dS) * dCT
    Rz = sin(dT)
    break
case 1:
    dS = (dU + 1) * PI
    dT = dV * dPi - dPi / 2

    Rx = cos(dS) * dCT
    Ry = sin(dS) * dCT
    Rz = sin(dT)
    break
case 2:
    dS = dU * PI
    dT = dV * dPi - dPi / 2

    Rx = sin(dS) * dCT
    Ry = sin(dT)
    Rz = cos(dS) * dCT
    break
case 3:
    dS = (dU + 1) * PI
    dT = dV * dPi - dPi / 2

    Rx = sin(dS) * dCT
    Ry = sin(dT)
    Rz = cos(dS) * dCT
    break
case 4:
    dS = dU * PI
    dT = dV * dPi - dPi / 2

    Rx = sin(dT)
    Ry = cos(dS) * dCT
    Rz = sin(dS) * dCT
    break
case 5:
    dS = (dU + 1) * PI
    dT = dV * dPi - dPi / 2

    Rx = sin(dT)
    Ry = cos(dS) * dCT
    Rz = sin(dS) * dCT
    break
```