Rendering Techniques Handout — Noise Functions

```
/*
 * Declarations and preprocessor definitions used in the various noise
 * functions.
 * Darwyn Peachey, June, 1994.
 */

#ifndef _NOISE_H_
#define _NOISE_H_ 1

#define TABSIZE         256
#define TABMASK         (TABSIZE-1)
#define PERM(x)         perm[(x)&TABMASK]
#define INDEX(ix,iy,iz)  PERM((ix)+PERM((iy)+PERM(iz)))

#define RANDMASK  0x7fffffff
#define RANDNBR   ((random() & RANDMASK)/(double) RANDMASK)

extern unsigned char perm[TABSIZE];         /* see perm.c */

extern float catrom2(float d);              /* see catrom2.c */

#endif /* _NOISE_H_ */
```

Value Noise

```
#include "proctext.h"
#include "noise.h"

static float valueTab[TABSIZE];

static void valueTabInit(int seed);
static float vlattice(int ix, int iy, int iz);

float
vnoise(float x, float y, float z)
{
   int ix, iy, iz;
   int i, j, k;
   float fx, fy, fz;
   float xknots[4], yknots[4], zknots[4];
   static int initialized = 0;
```

```
    if (!initialized) {
        valueTabInit(665);
        initialized = 1;
    }

    ix = FLOOR(x);
    fx = x - ix;

    iy = FLOOR(y);
    fy = y - iy;

    iz = FLOOR(z);
    fz = z - iz;

    for (k = -1; k <= 2; k++) {
        for (j = -1; j <= 2; j++) {
            for (i = -1; i <= 2; i++)
                xknots[i+1] = vlattice(ix+i,iy+j,iz+k);
            yknots[j+1] = spline(fx, 4, xknots);
        }
        zknots[k+1] = spline(fy, 4, yknots);
    }
    return spline(fz, 4, zknots);
}

static void
valueTabInit(int seed)
{
    float *table = valueTab;
    int i;

    srandom(seed);
    for(i = 0; i < TABSIZE; i++)
        *table++ = 1. - 2.*RANDNBR;
}

static float
vlattice(int ix, int iy, int iz)
{
    return valueTab[INDEX(ix,iy,iz)];
}
```

Gradient Noise

```
#include "proctext.h"
#include "noise.h"

#define SMOOTHSTEP(x)  ((x)*(x)*(3 - 2*(x)))

static float gradientTab[TABSIZE*3];

static void gradientTabInit(int seed);
static float glattice(int ix, int iy, int iz, float fx, float fy, float fz);

float
gnoise(float x, float y, float z)
{
   int ix, iy, iz;
   float fx0, fx1, fy0, fy1, fz0, fz1;
   float wx, wy, wz;
   float vx0, vx1, vy0, vy1, vz0, vz1;
   static int initialized = 0;

   if (!initialized) {
      gradientTabInit(665);
      initialized = 1;
   }

   ix = FLOOR(x);
   fx0 = x - ix;
   fx1 = fx0 - 1;
   wx = SMOOTHSTEP(fx0);

   iy = FLOOR(y);
   fy0 = y - iy;
   fy1 = fy0 - 1;
   wy = SMOOTHSTEP(fy0);

   iz = FLOOR(z);
   fz0 = z - iz;
   fz1 = fz0 - 1;
   wz = SMOOTHSTEP(fz0);

   vx0 = glattice(ix,iy,iz,fx0,fy0,fz0);
   vx1 = glattice(ix+1,iy,iz,fx1,fy0,fz0);
   vy0 = LERP(wx, vx0, vx1);
   vx0 = glattice(ix,iy+1,iz,fx0,fy1,fz0);
```

```
   vx1 = glattice(ix+1,iy+1,iz,fx1,fy1,fz0);
   vy1 = LERP(wx, vx0, vx1);
   vz0 = LERP(wy, vy0, vy1);

   vx0 = glattice(ix,iy,iz+1,fx0,fy0,fz1);
   vx1 = glattice(ix+1,iy,iz+1,fx1,fy0,fz1);
   vy0 = LERP(wx, vx0, vx1);
   vx0 = glattice(ix,iy+1,iz+1,fx0,fy1,fz1);
   vx1 = glattice(ix+1,iy+1,iz+1,fx1,fy1,fz1);
   vy1 = LERP(wx, vx0, vx1);
   vz1 = LERP(wy, vy0, vy1);

   return LERP(wz, vz0, vz1);
}

static void
gradientTabInit(int seed)
{
   float *table = gradientTab;
   float z, r, theta;
   int i;

   srandom(seed);
   for(i = 0; i < TABSIZE; i++) {
      z = 1. - 2.*RANDNBR;
      /* r is radius of x,y circle */
      r = sqrtf(1 - z*z);
      /* theta is angle in (x,y) */
      theta = 2 * M_PI * RANDNBR;
      *table++ = r * cosf(theta);
      *table++ = r * sinf(theta);
      *table++ = z;
   }
}

static float
glattice(int ix, int iy, int iz,
   float fx, float fy, float fz)
{
   float *g = &gradientTab[INDEX(ix,iy,iz)*3];
   return g[0]*fx + g[1]*fy + g[2]*fz;
}
```