JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 7, NO. 12, DECEMBER 2004(pp. 1700-1707)

Combination Algorithm of a Material for Marble Solid Effects

Tae-Jin Park[†], Man-Gon Park[††]

ABSTRACT

Nowaday, market size of digital image in world around is looks to rapidly growth. For this, Texture mapping has traditionally been used to add realism to computer graphics images. Therefore to make our image realistic, we need to give the various kind of objects material parameter and environment lighting. To present the completed marble we use passing back algorithm and combination with channel of a material. In experimental result of this paper that application by passing back algorithm and varying the parameter such as scale, period, distortion, octaves of noise make showing the superiority of optimized rendering of spheres and perfect another marble effects.

Keywords : Solid marble texture, perlin noise, turbulence function, intersection 3D sphere, passing back algorithm

1. INSTRUCTION

Nowaday, market size of digital image in world around is looks to rapidly growth. For this, Texture mapping has traditionally been used to add realism to computer graphics images. Texture mapping [1,4] is a powerful technique for adding realism to a computer-generated scene. Therefore we used Perlin Noise to develop procedural marble textures for any point in 3D space. The idea of a procedural texture map is simply that for any point in 3D space, you get back a number (noise function) between -1 and 1. From here, we used the sin function along with parameter to control the octaves, frequency, scale, and distortion of noise function.

※ Corresponding Author : Tae-Jin Park, Address : (608-737) 599-1 Daeyon-dong, Nam-gu, Busan, Korea, TEL : +82-51-620-6391, FAX : +82-51-628-6155
E-mail : csptj@mail1.pknu.ac.kr
Receipt date: Oct. 7, 2004, Approval date: Dec. 16, 2004
[††] Division of Computers & Multimedia Engineering, Pukyong Nat'l Univ.
[††] Division of Computers & Multimedia Engineering, Pukyong Nat'l Univ.
(E-mail : mpark@pknu.ac.kr)

In this paper application by passing back algorithm and varying the parameter such as a scale, period, distortion, octaves of noise was created in order to optimized rendering of spheres and different marble effects. Basically, if we can cut down the amount of intersection tests per ray then we could dramatically increase the speed of rendering.

2. BACKGROUND AND PREVIOUS WORK

2.1 Dimensional

A common technique is to create 1/fn noise which is known to occur often in natural processes. An approximation to this is to add suitably scaled harmonics of this basic noise function. For the rest of this discussion the Perlin noise functions[2] will be referred to as N(x) of variable x which may a vector in 1, 2, 3 or higher dimension. This function will return a real (scalar) value. A harmonic will be N (b x) where 'b' is some positive number greater than 1, most commonly it will be power of 2. While the N() functions can be used by themselves, a more common approach is to create a weighted sum of a number of harmonics of these functions. These will be referred to as N(x) and can be defined as

$$n_{(x)} = \sum_{i=0}^{N-1} \frac{n(b^i x)}{a^i}$$

Where N is typically between 6 and 10. The parameter ‘a’ controls how rough the final N() function will be. Small values of ‘a’, give very rough functions, larger values give smoother functions. While this is the standard form in practice it isn’t uncommon for the terms a_i and b_i to be replaced by arbitrary values for each i.

2.2 Solid Noise

To design n, just calling a random number for every point wouldn’t be appropriate because it would just be like “white noise” in TV static. We would like to make it smoother without losing the random quality. One possibility would be to blur white noise, but there is no practical implementation of this that can be evaluated in isolation. Another possibility would be to make a big lattice with a stored random number at every Cartesian lattice point (x, y, and z all integers) and interpolate these random points for points between lattice nodes. This would make lattice too obvious, and you would need too much memory.

Perlin introduced a modification of this basic idea that both hides the lattice and requires little memory. His function n (p) is usually called Perlin noise. Perlin used a variety of tricks to improve this basic lattice technique[2,3]. First, he placed randomly-chosen vectors at the lattice points and used dot products to move the extrema off the lattice. Second, he used higher-order interpolation rather than trilinear to hide derivative artifacts. Finally, he used hashing so that he could use a small table for all the lattice points. Here is his basic method :

$$n(x,y,z) = \sum_{i=\lfloor x\rfloor}^{\lfloor x\rfloor+1} \sum_{j=\lfloor y\rfloor}^{\lfloor y\rfloor+1} \sum_{k=\lfloor z\rfloor}^{\lfloor z\rfloor+1} \Omega_{(ijk)}(x-i,\ y-j,\ z-k)$$

where (x,y,z) are the Cartesian coordinates of x, and

$$\Omega_{(ijk)}(u,v,w) = \omega(u)\,\omega(v)\,\omega(w)\,(g_{ijk}\cdot(u,v,w)),$$

and $\omega(t)$ is the cubic weighting functions :

$$\omega(t) = \begin{cases} -6|t|^6 + 15|t|^4 - 10|t|^3 + 1 & \text{if } |t| < 1, \\ 0 & \textit{otherwise.} \end{cases}$$

The final piece is that g_{ijk} is a vector for the lattice point (x,y,z) = (i,j,k). Since we want any potential i,j,k, we use a pseudorandom table :

$$g_{i,j,k} = G(\Phi(i+\Phi(j+\Phi(k))))$$

where G is a precomputed array of N vectors, and $\Phi(i) = P[i \bmod N]$,

P is an array of length N containing a permutation of the integers 0 through N-1.

Perlin suggests N=16 and the following vector :

(1, 1, 0)	(1, 0, 1)	(-1, 1, 0)	(-1, 0, 1)
(1,-1, 0)	(1, 0,-1)	(-1,-1, 0)	(-1, 0,-1)
(0, 1, 1)	(1, 1, 0)	(0,-1, 1)	(-1, 1, 0)
(0, 1,-1)	(0,-1, 1)	(0,-1,-1)	(0,-1,-1)

The first twelve vectors are from the origin to the twelve edges of a cube, and the next four are padding so that mod can be efficient. Because solid noise can be positive or negative, it must be transformed before being converted to a color.

The dark curves are where the original noise function went from positive to negative. Since noise varies from -1 to 1, a smoother image can be achieved by using (n(p)+1)/2 for color. However, since noise values close to 1 or -1 are rare, this will be a fairly smooth image. Larger scaling can increase the contrast.

2.3 Turbulence

Many natural textures contain a variety of feature sizes in the same texture.

Perlin uses a pseudofractal “turbulence” function :

$$n_{t(p)} = \sum_{i}^{M} \frac{n(2^i p)}{2^i}$$

The turbulence function is shown of resulting value differently according to various values of M.

1) Application of Perlin's function for 3d noise and turbulence

All for images [Fig. 1] as above use Perlin's function for 3d noise and turbulence[2]. Perlin's function is used to calculate the value of "t" in the following equation :

float c = 0.5×(sin((period×point.x()) + (t×distortion)) + 1.0)

The value of "c" is then used to create RGB values for the spheres. The period in all cases is 20.0. The distortion is the only variable in these images.

2) Other function using Perlin's noise

As shown images [Fig. 2], (e) to (h) have been generated using Perlin's noise function differently than images (a) to (d). Perlin's function is used to calculate the value of "t" in the following equatio n : float c = cos(point.x() + t)

The value of "c" is used to create RGB values for the spheres.

3. IMPLEMENTATION

A marble-like effect is created using a solid texture generated by a noise function from Ken Perlin. Different marble effects are possible by varying the parameters : scale, period, distortion, and octaves, which are input file arguments that define marble textures.

3.1 Passing Back Algorithm

We present a virtual texture class for storing the color attributes for primitives. One implementational question that arises is where to calculate the color of the intersected point on the hit object. We could have the object calculate its color within the hit function and then pass it back in the hit record as we have done before.

In this [Fig. 3] as follows, the shadowHit routine of defined class Shape is a simple efficiency technique that takes advantage of the fact that we don't need to know any information about the occluding object. In the regular hit function we would like to pass back some information about the hit object. In order to keep the number of parameters to the hit function reasonable, we use a struct to pass back this information[5].

We pass two parameters into the virtual value function. The Vector3 parameter is 3D hit point we

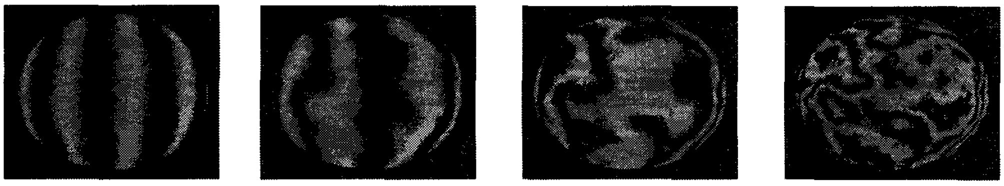

(a) distortion=1 (b) distortion=10 (c) distortion=20 (d) distortion=40

Fig. 1. Using Perlin's function and then various turbulent sphere by distortion.

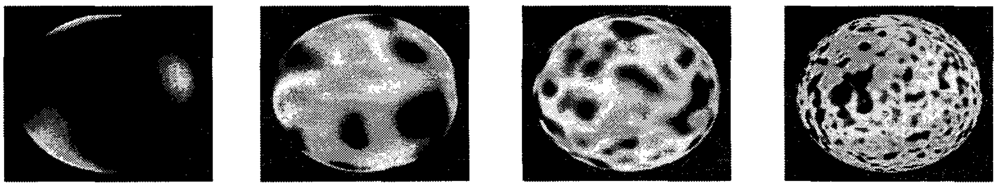

(e) distortion=1 (f) distortion=10 (g) distortion=20 (h) distortion=40

Fig. 2. Using Perlin's other function and then various turbulent sphere by distortion.

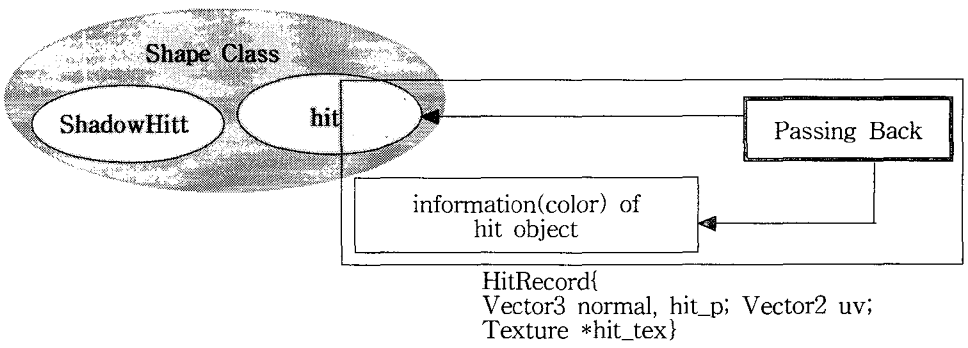

Shape Class
ShadowHitt
hit
Passing Back
information(color) of hit object
HitRecord{
Vector3 normal, hit_p; Vector2 uv;
Texture *hit_tex}

Fig. 3. Passing Back Algorithm.

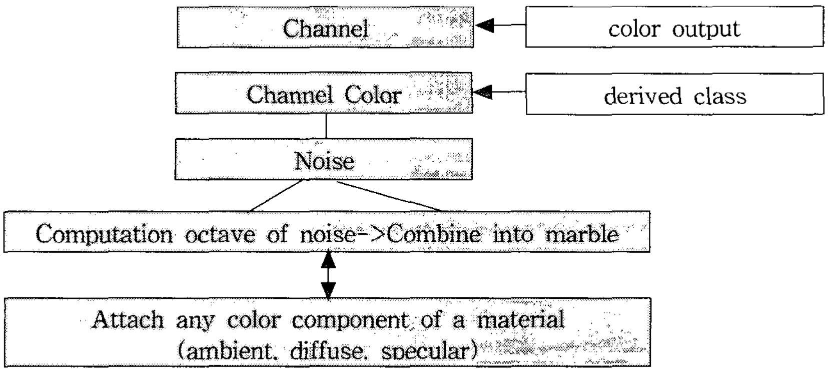

Channel
color output
Channel Color
derived class
Noise
Computation octave of noise->Combine into marble
Attach any color component of a material (ambient, diffuse, specular)

Fig. 4. Channel class for attach component of a material into marble.

```
material "red" {
    diffuse { color < 1 0 0 > }
}
```
which will set the diffuse component to a constant red.

```
material "marble" {
     diffuse {
         marble {
           scale 1.0
             period 1.3
             distortion 3
             octaves 12
             ramp {
           < 1.0 0.000 0.0 >
                        < 1.0 0.125 0.0 >
                        < 1.0 0.250 0.0 >
                        < 1.0 0.375 0.0 >
                        < 1.0 0.500 0.0 >
                        < 1.0 0.625 0.0 >
                        < 1.0 0.750 0.0 >
                        < 1.0 0.875 0.0 >
           }
         }
     }
}
```

Fig. 5. Code for setting diffuse channel to the marble.

```
float n = 0.0f;
for ( int i = 0; i < octaves; i++ )
{  float o = ( float )( 1 << i );
   n += Noise3( v * o ) / o; }
return n;
```

Fig. 6. Code for computing octaves of noise.

shading(hit_p in the hit record). The Vector2 parameter will be used in implement 2D texture maps. Also, vector class is one of the foundations of most graphics programs. We have used aggressive inline to avoid function call overhead and use of const functions allow the compiler to more aggressively optimize. There are other techniques for speeding up a vector class which involve added data members, such as length, which are precomputed during initialization.

3.2 Computation Octaves of Noise

Basically as you know Perlin noise simply takes a sum of a set of noise functions, multiplying at each s uccessive octave passed into the noise function by 2^i, and then the result is divided by 2^i. Note that an octave corresponds to an increase by a factor of 2 in the frequency, which is exactly what happens when you add successive Perlin components together.

The sine function for computing the new turbstripe function looks as follows :

$$t = fabs(sin(3.1415926535 \times (frequency \times p.y + distortion \times PerlinNoise(p \times scale,\ octaves)\ )\ ));$$

The PerlinNoise function is simply an iterative function that does the previously described summation of noise.

We first created a new class called a *Channel. The* function of class is to provide a color output. There is a class derived from the channel called a *ChannelColor*, which is simply a constant color. Derived from the *Channel* class is also a *Channel-Marble* class which instantiates a *Noise* class to compute octaves of noise and combines them into marble. In the scene description file, you can now attach any color component of a material (currently only ambient, diffuse and specular)[1] to any of these channels. It looks like this in the scene file :

which will set the diffuse channel to the marble function with the given parameters as above [Fig. 5]. The function, We use to compute the marble looks something like the following :

$$abs(sin(180 \times (period \times v.x + distortion \times Fractal3(v,octaves)\ )\ );$$

where v is the (possibly scaled) position at which the noise is to be computed and *Fractal3* is a fractal sum of noise computed as follows :

Where *Noise3* is the standard Perlin Noise function.

1) material called ambient, diffuse, and specular term was have relation with lighting that was added to the color every surface.

4. EXPERIMENTAL RESULTS

The measurements presented in this paper were using a Pentium(R) 4-M CPU 2.00 GHZ with 512MB SDRAM and return the elapsed time in the form (HHhMMmSS.Ss) are for 500×500 pixel images.

To present the completed marble, we use an passing back algorithm and combination with channel of a material. In experimental result of this paper that application by passing back algorithm and varying the parameter such as scale, period, dis-tortion, octaves of noise make showing the superiority of optimized rendering of spheres and perfect another marble effects.

Experimental method of this paper is trying to optimize a ray tracer for the sort of sphere objects then all of the images above can be regenerated by specifying the appropriate scene file on the command line of the program.

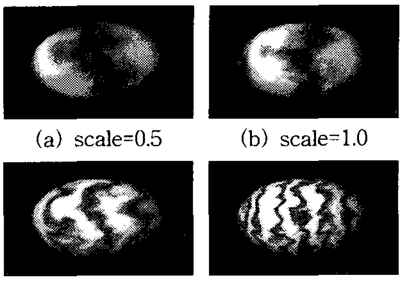
As you looks our experimental data above that each (a), (b), (c), (d) image in [Fig. 7-10] result of render can presented another effect according to change value of each noise parameter. Moreover, We have to found that experimental result value of render are showing to give effected by octaves of noise. Also experimentation 5 of [Fig. 11] that have to make a best realistic looking marble sphere.

5. CONCLUSIONS AND FUTURE WORK

Nowaday, market size of digital image in world around is looks to rapidly growth. For this, not only but also generation solid texture that we can create various kind of realistic looking marble texture.

As referred in this study, application by passing back algorithm and varying the parameter such as scale, period, distortion, octaves of noise make showing the superiority of optimized rendering of

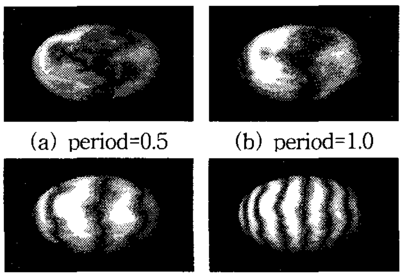
	scale	period	distortion	octaves	render
1	**0.5**	1.0	1.0	8	2.0
2	**1.0**	1.0	1.0	8	2.0
3	**2.0**	1.0	1.0	8	2.0
4	**4.0**	1.0	1.0	8	2.0



marble solid texture at different scale

Fig. 7. Experimentation 1 : different scale.

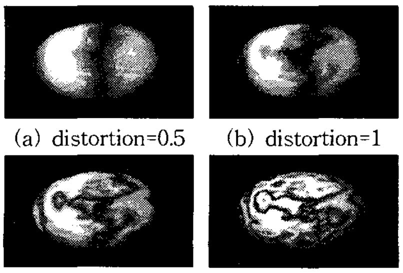
	scale	**period**	distortion	octaves	render
1	1.0	**0.5**	1.0	8	2.0
2	1.0	**1.0**	1.0	8	2.0
3	1.0	**2.0**	1.0	8	2.0
4	1.0	**4.0**	1.0	8	2.0



marble solid texture at different periods

Fig. 8. experimentation 2 : different period.

	scale	period	**distortion**	octaves	render
1	1.0	1.0	**0.5**	8	2.0
2	1.0	1.0	**1.0**	8	2.0
c3	1.0	1.0	**2.0**	8	2.0
4	1.0	1.0	**4.0**	8	2.0



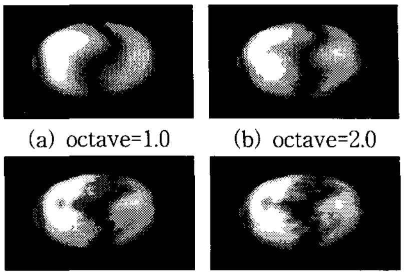
marble solid texture at different distortion

Fig. 9. experimentation 3 : different distortion.

spheres and perfect another marble effects. Basically, if we can cut down the amount of intersection tests per ray then we could dramatically increase the speed of rendering.

We have presented a method that further work will include the ability to combine several channels arithmetically, so We can get multi-pass effects, or for things such as a base diffuse color modulated by a marble procedure or an image (texture) map.

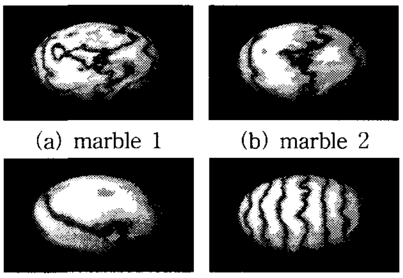
	scale	period	distortion	**octaves**	render
1	1.0	1.0	1.0	**1**	0.8
2	1.0	1.0	1.0	**2**	1.0
3	1.0	1.0	1.0	**4**	1.4
4	1.0	1.0	1.0	**8**	2.0



marble solid texture at different octaves

Fig. 10. experimentation 4 : different octave.

	scale	period	distortion	octaves	render
1	1.0	1.3	3.0	12	2.8
2	1.0	1.5	1.7	32	6.3
3	0.3	1.2	3.0	8	2.1
4	2.7	1.6	0.6	8	2.1



best attempts at making realistic looking marble sphere

Fig. 11. experimentation 5 : realistic looking marble sphere.

6. REFERENCES

[1] P. S. Heckbert, "A Survey of Texture Mapping", IEEE Computer Graphics & Applications, pp. 56-67, 1986.

[2] Perline and Eric Hoffert, "Hypertexture", Computer Graphics, Proceedings of ACM SIGGRAPH, '89, Vol. 23, No. 3, 1989.

[3] Perline, "Improving Noise", Transaction on Computer Graphics, Proceedings of ACM SIGGRAPH '02, Vol. 21, No. 3, pp. 0681-0682, 2002.

[4] David S. Ebert, Texturing & Modeling : A Procedural Approach, 3rd edition, Morgan Kaufmann Publishers, San Francisco, 2002.

[5] Upstill, The Renderman Companion : A Programmer's Guide to Realistic Computer Graphics, Addison-Wesley Publishers, 1990.

[6] Ron Brinkmann, The Art and Science of Digital Composition, Academic Press, San Diego, pp. 1-121, 2001.

[7] Anthony.A Apodaca, Larry Gritz, Advanced Renderman : Creating CGI for Motion Pictures, Morgan Kaufmann Publishers, pp. 20-70, 2001.

Tae-Jin Park

He received his B.S. degree in physics from Dongeui University, Busan, in 1988. He received the M.S degree in Computer Science and Information from Pukyong National University, Busan in 1995. And he received his Completion of a Doctor's course in computer science from Pukyong National University, Busan in 2002.

He worked for Kyungnam College of Information as a Lecturer from 1995 to 1997. And he worked for Dongeui Institute of Technology as concurrent professor in Dept. of Electrical and Computer Engineering. And also worked the Grouping of Electrical and Computer Engineering of Koje College as a Invitation professor from 2000 to 2004.

His main interests are in multimedia information system, computer graphics, medical image, embedded system, software reliability and safety engineering.

Man-Gon Park

1976 B.S, Dept. of Mathematics Education, KyungPook National University
1980 M.S, Dept. of Computing Science & Education, KyungPook National Univ.
1987 Ph.D, Dept. of Computing Science & Education, KyungPook National Univ.
1992~1993 Post Doctoral Course in Computer Engineering, Dept. of Electrical & Computer Engineering, Univ. of Kansas, Lawrence, USA
1979~1981 Professor, Dept. of Computer Science, KyoungNam Technical College
1987~1990 Director of Central Computer Center/ Vice Dean of Nature Science College, PuKyong National University (PKNU)
1988~1997 Chairman of Dept. of Computer Science (Undergraduate & Graduagte Courses), PKNU
1997~2002 Faculty Consultant, Div. of Information Technology & Communication, Colombo Plan Staff College, Manila, Philippines
1981~Present Professor, Dept. of Computer Science, PKNU
1997~Present Advisory Professor for International Information Technology, ADB/ UN ESCAP/ ILO APSDEP/ KOICA/ JICA
2002~Present Director, Colombo Plan Staff College (Intergovernmental International Organization), Manila, Philippines
Research Interests : Software Reliability & Safety Engineering, Software Quality Engineering, Software Metrics, Software Reusability & Reengineering, Software Testing & Inspection, Fault-Tolerant Software System, Methodology of Information System Development, Internet, BPR, GIS, and Multimedia Information Processing Techinques