OSL in Blender

- Success story
- Story of artist involvement in contributing plugins
- Example: New procedural textures
- Even completely new tools are already available this way
- You can get out-of-the-box solutions or small building blocks

Past:

```
#include "math.h"
#include "plugin.h"

#define NR_TYPES 1


float result[8];
float cfra;

intdo_reset = FALSE;

extern float hnoise(float noisesize, float x, float y, float z);

/* set up plugin menu */

char name[]= "Pie";
char stnames[NR_TYPES][16]= {"Pie"};

VarStruct varstr[]= {
/* type, name,  default,min,      max,    tooltips */
{ LABEL,  "", 0,     0, 0, ""},
{ NUM|INT,   "divs ", 6.0,  2.0,  1000.0,
"Pie plugin: Number of slices"},
/* Number of slices */
{ NUM|FLO, "hardness ", 1.0,  0.0,  5.0,
"Pie plugin: Falloff Hardness"},
/* determins sharpnes of edge, could use some work */
{ NUM|FLO, "ang ofs ", 0.0, -180.0, 180.0,
"Pie plugin: Angle offset "},
/* phase angle */
{ NUM|FLO,     "turb dep ",0.0,    -5.0,  5.0,
"Pie plugin: Turbulance depth"},
/* Pos. Turb affects 'white', Neg. affects 'black' */
{ NUM|FLO, "turb siz ",0.25,    0.0,  2.0,
"Pie plugin: Turbulance size"}
};
typedef struct Cast {
float dum1;
int div;
float hard;
float ang;
float turbd;
float turbs;
} Cast;

/* ****************************************************************** */

int plugin_tex_doit(int, Cast*, float*, float*, float*);

int plugin_tex_getversion(void)
{
return B_PLUGIN_VERSION;
}

void plugin_but_changed(int but)
{
}

void plugin_init(void)
{
}

void plugin_getinfo(PluginInfo *info)
{
info->name= name;
info->stypes= NR_TYPES;
info->nvars= sizeof(varstr)/sizeof(VarStruct);

info->snames= stnames[0];
info->result= result;
info->cfra= &cfra;
info->varstr= varstr;

info->init= plugin_init;
info->tex_doit=  (TexDoit) plugin_tex_doit;
info->callback= plugin_but_changed;
}

int plugin_tex_doit(int stype, Cast *cast, float *texvec, float *dxt, float *dyt)
{
float angle, turb=0;

angle = atan2(texvec[0],texvec[1]) + cast->ang*3.1415926/180.0;

if ( cast->turbd !=0.0 )  {     /* save time if no turb */
turb = cast->turbd * hnoise(cast->turbs,texvec[0],texvec[1], texvec[2]);
/*printf("turb: %f\n",turb);*/
}

result[0]= 0.5 - 0.5*sin(angle * cast->div - turb - 0.5);
if (cast->hard !=1) result[0]= pow(result[0], cast->hard);  /*Very slow, better way?*/

    if (result[0] > 1) result[0] = 1.0;
else if (result[0]<0) result[0]=0;

result[4]= 1.0;

return 0;
}
```

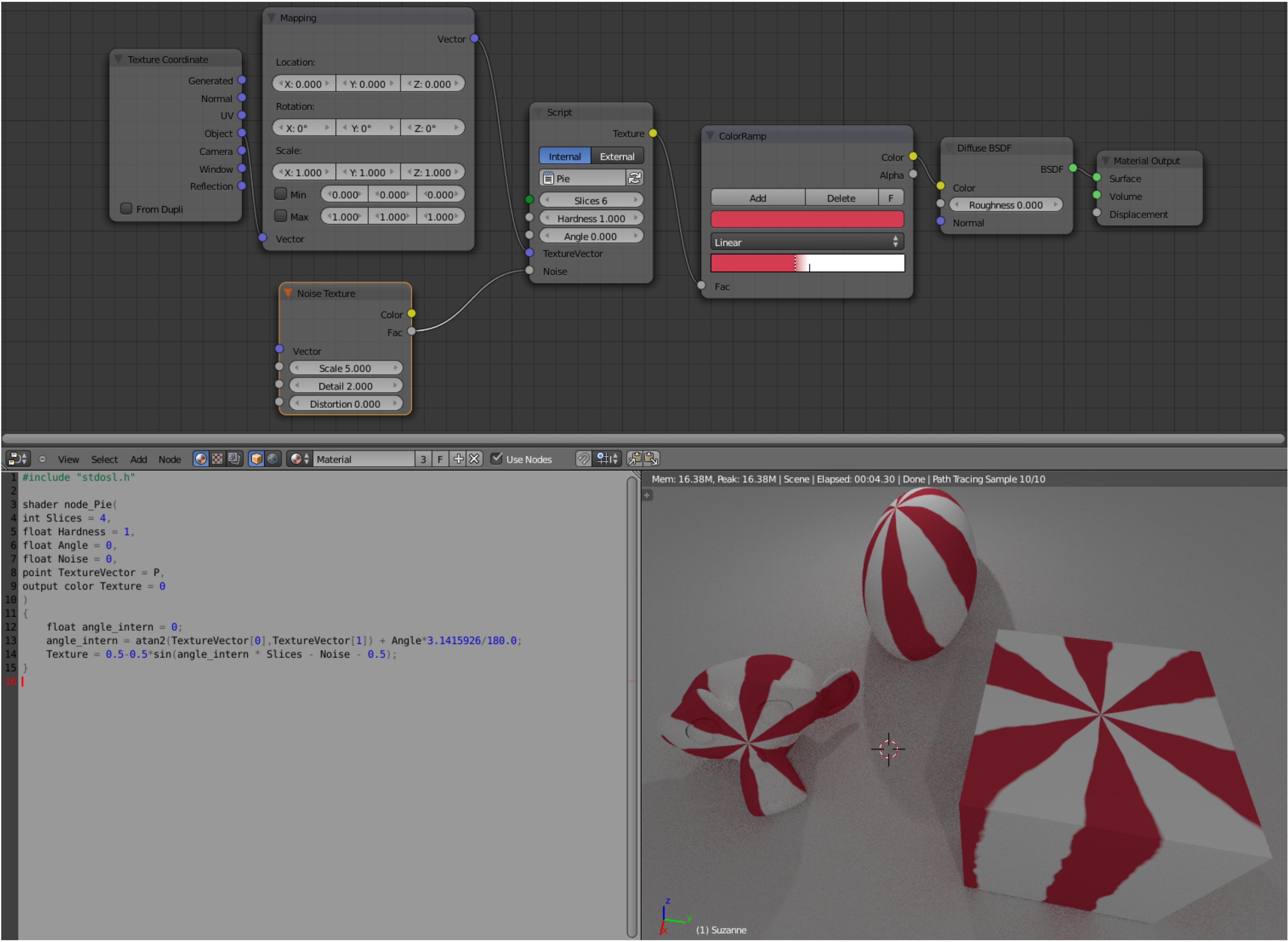
Present:

```
#include "stdosl.h"


shader node_Pie(
int Slices = 4,
float Angle = 0,
float SmoothCenter = 0,
float Noise = 0,
point TextureVector = P,
output color Texture = 0
)
{
   float angle_intern = 0;
   float centerDot = 0;

   centerDot =
sqrt(TextureVector[0]*TextureVector[0]+TextureVector[1]*TextureVector[1]);
   centerDot = 1-centerDot;
   centerDot = smoothstep(1-SmoothCenter,1,centerDot);
   angle_intern = atan2(TextureVector[0],TextureVector[1]);
   angle_intern -= radians(Angle);
   float piepattern = 0.5-0.5*sin(angle_intern * Slices - Noise );
   piepattern += centerDot;
   piepattern = clamp(piepattern, 0,1);

   Texture = piepattern;
}
```



Mapping
Location: X: 0.000 Y: 0.000 Z: 0.000
Rotation: X: 0° Y: 0° Z: 0°
Scale: X: 1.000 Y: 1.000 Z: 1.000
□ Min 0.000 0.000 0.000
□ Max 1.000 1.000 1.000

Script
Pie
Slices 6
Hardness 1.000
Angle 0.000

ColorRamp
Linear

Diffuse BSDF
Roughness 0.000

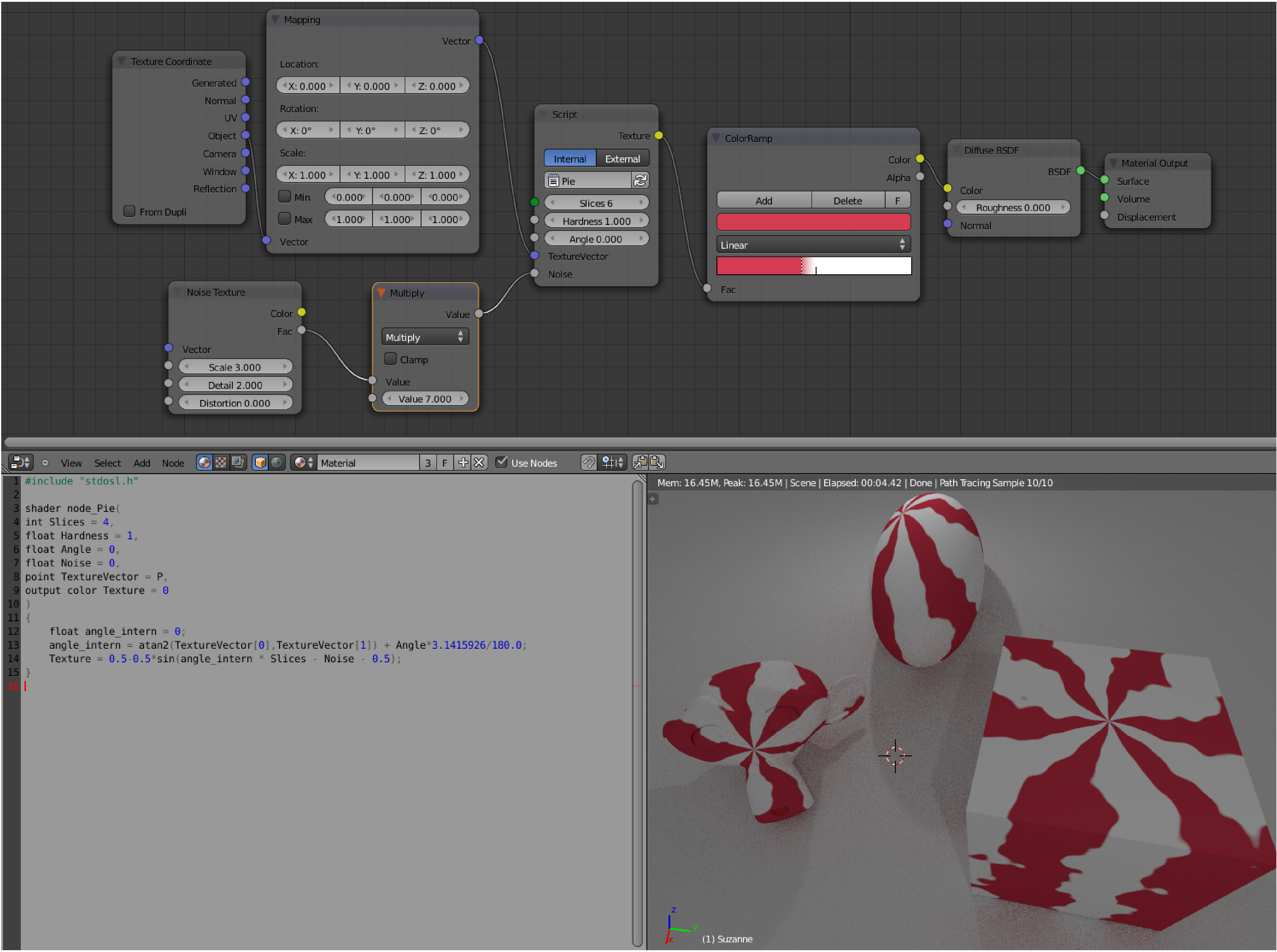
Noise Texture
Scale 5.000
Detail 2.000
Distortion 0.000

Material 3 F

```
#include "stdosl.h"

shader node_Pie(
int Slices = 4,
float Hardness = 1,
float Angle = 0,
float Noise = 0,
point TextureVector = P,
output color Texture = 0
)
{
    float angle_intern = 0;
    angle_intern = atan2(TextureVector[0],TextureVector[1]) + Angle*3.1415926/180.0;
    Texture = 0.5-0.5*sin(angle_intern * Slices - Noise - 0.5);
}
```

Mem: 16.38M, Peak: 16.38M | Scene | Elapsed: 00:04.30 | Done | Path Tracing Sample 10/10



Texture Coordinate: Generated, Normal, UV, Object, Camera, Window, Reflection, □ From Dupli

Mapping: Vector; Location: X: 0.000, Y: 0.000, Z: 0.000; Rotation: X: 0°, Y: 0°, Z: 0°; Scale: X: 1.000, Y: 1.000, Z: 1.000; □ Min 0.000 0.000 0.000; □ Max 1.000 1.000 1.000; Vector

Script: Texture; Internal / External; Pie; Slices 6; Hardness 1.000; Angle 0.000; TextureVector; Noise

ColorRamp: Color; Alpha; Add, Delete, F; Linear; Fac

Diffuse BSDF: BSDF; Color; Roughness 0.000; Normal

Material Output: Surface; Volume; Displacement

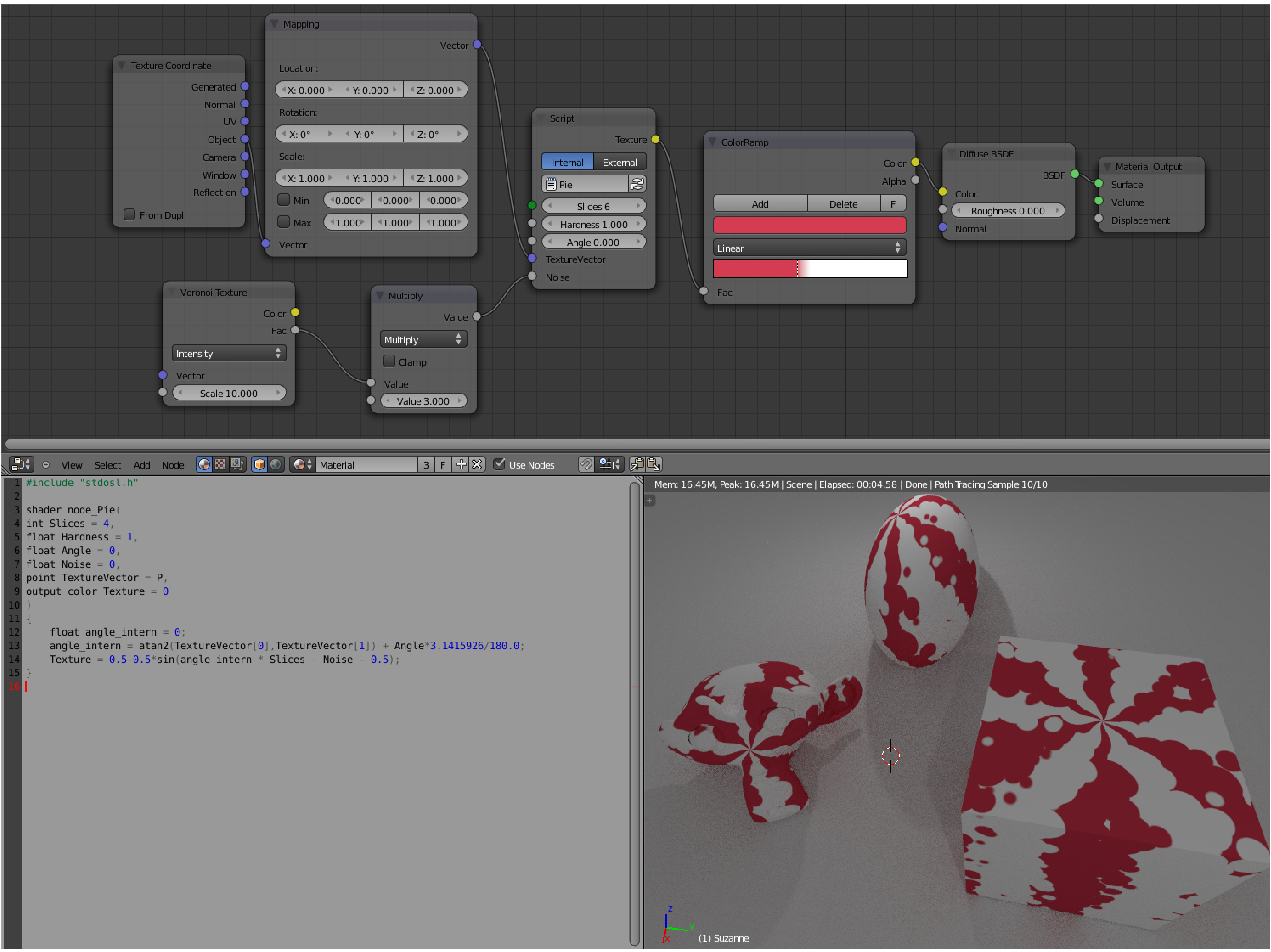
Noise Texture: Color; Fac; Vector; Scale 3.000; Detail 2.000; Distortion 0.000

Multiply: Value; Multiply; □ Clamp; Value; Value 7.000

```
#include "stdosl.h"

shader node_Pie(
int Slices = 4,
float Hardness = 1,
float Angle = 0,
float Noise = 0,
point TextureVector = P,
output color Texture = 0
)
{
    float angle_intern = 0;
    angle_intern = atan2(TextureVector[0],TextureVector[1]) + Angle*3.1415926/180.0;
    Texture = 0.5-0.5*sin(angle_intern * Slices - Noise - 0.5);
}
```

```
Mem: 16.45M, Peak: 16.45M | Scene | Elapsed: 00:04.42 | Done | Path Tracing Sample 10/10
```



Mapping — Location: X: 0.000 Y: 0.000 Z: 0.000

Rotation: X: 0° Y: 0° Z: 0°

Scale: X: 1.000 Y: 1.000 Z: 1.000

□ Min 0.000 0.000 0.000

□ Max 1.000 1.000 1.000

Voronoi Texture — Intensity; Scale 10.000

Multiply — Multiply; □ Clamp; Value 3.000

Script — Pie; Slices 6; Hardness 1.000; Angle 0.000

ColorRamp — Linear

Diffuse BSDF — Roughness 0.000

☑ Use Nodes

```
#include "stdosl.h"

shader node_Pie(
int Slices = 4,
float Hardness = 1,
float Angle = 0,
float Noise = 0,
point TextureVector = P,
output color Texture = 0
)
{
    float angle_intern = 0;
    angle_intern = atan2(TextureVector[0],TextureVector[1]) + Angle*3.1415926/180.0;
    Texture = 0.5-0.5*sin(angle_intern * Slices - Noise - 0.5);
}
```

Mem: 16.45M, Peak: 16.45M | Scene | Elapsed: 00:04.58 | Done | Path Tracing Sample 10/10

Past

- Few procedural textures to prevent feature-creep
- Only nodes and tools the core developers offered to the users

Present

- Myriads of procedural textures and other custom shaders available as plugins
- Ability to customize nodes to the user's expectations
- All without interfering with the Blender core development

Why OSL was a success

- Integrates well with Blender's node-based shader system
- Semantically similar to Cycles node
- Syntax close to RSL, GLSL, HSL etc.
- No Boilerplate code whatsoever
- (Nearly) Live-coding possible

OSL Resources

- http://www.openshading.com/

  Tutorials and News on OSL and Cycles by Thomas Dinges

- https://www.smashwords.com/books/view/368598

  Open Shading Language for Blender – A Practical Primer by Michael Anders

  (Must-read if you want to dive into OSL coding, includes lots of finished shaders)

- https://github.com/sambler/osl-shaders

  Huge collection of OSL shaders that are ready to use

- https://github.com/GottfriedHofmann/osl-lib

  WIP of small and useful tools and procedural textures for Cycles

- http://blenderartists.org/forum/forumdisplay.php?47-Coding

  Has two sections dedicated to OSL, you will find useful shaders in both

- http://cgcookie.com/blender/cgc-courses/introduction-to-osl-in-blender-cycles/

  OSL coding intro

- http://blendersushi.blogspot.de/

  Follow Jimmy Gunawan on his journey through OSL