Animating with Renderman Shading Language
Responsible for everything. 2015.

https://youtu.be/bspbSeG37-o

ST Coloration:
These images show the results of using the RenderMan Shading Language (RSL) to write a variety of traditional surface shaders. The notes and RSL code accompanying each image explain how each pattern was generated.

```
/* shader_src/circlequarter.sl */
surface
circlequarter(  float  Kfb = 1;
   float s_center = 0;
   float t_center = 0;
   color backcolor_a = color(0,1,0);
   color backcolor_b = color(0,1,1);
   color forecolor_a = color(1,1,0);
   color forecolor_b = color(1,0,0);
   float fore_direction = 1 /* [0 or 1] */;
   float back_direction = 1 /* [0 or 1] */;

   )



{

color  surfcolor = 1;
float d=distance (point(s,t,0),point(s_center,t_center,0));

   if(d <= 1){
    if(fore_direction == 1)
    surfcolor =  mix(forecolor_a, forecolor_b, s);
    else
    surfcolor =  mix(forecolor_a, forecolor_b, t);
    }

   else{
    if(back_direction == 1)
    surfcolor = mix(backcolor_a, backcolor_b, s );
    else
    surfcolor = mix(backcolor_a, backcolor_b, t );
   }

Oi = Os;

/* STEP 2 - calculate the apparent surface color */
Ci = Oi * Cs * surfcolor * Kfb;
}
```

```
/* shader_src/moon.sl */
surface
moon(  float  Kfb = 1;
   float big_s_center = 0.5;
   float big_t_center = 0.5;
   float big_radius = 0.3;
   float small_s_center = 0.5;
   float small_t_center = 0.3;
   float small_radius = 0.2;
   color backcolor_a = color(0,1,0);
   color backcolor_b = color(0,1,1);
   color forecolor_a = color(1,1,0);
   color forecolor_b = color(1,0,0);
   float fore_direction = 1 /* [0 or 1] */;
   float back_direction = 1 /* [0 or 1] */)
{
color  surfcolor;

if(back_direction == 1)
surfcolor = mix(backcolor_a, backcolor_b, s );
else
surfcolor = mix(backcolor_a, backcolor_b, t );

float big_dis=distance (point(s,t,0),point(big_s_center,big_t_center,0));


if(big_dis <= big_radius){

   if(fore_direction == 1)
   surfcolor =  mix(forecolor_a, forecolor_b, s);
   else
   surfcolor =  mix(forecolor_a, forecolor_b, t);

}


float small_dis=distance (point(s,t,0),point(small_s_center,small_t_center,0));


if(small_dis <= small_radius){


  if(back_direction == 1)
  surfcolor = mix(backcolor_a, backcolor_b, s );
  else
  surfcolor = mix(backcolor_a, backcolor_b, t );

}
```

```
Oi = Os;

/* STEP 2 - calculate the apparent surface color */
Ci = Oi * Cs * surfcolor * Kfb;
}
```

```
/* shader_src/diagonal.sl */
surface
diagonal(  float  Kfb = 1;
     float flip = 0 /* [0 or 1] */;
     color backcolor_a = color(0,1,0);
     color backcolor_b = color(0,1,1);
     color forecolor_a = color(1,1,0);
     color forecolor_b = color(1,0,0);
     float fore_direction = 1 /* [0 or 1] */;
     float back_direction = 1 /* [0 or 1] */

     )
{
color  surfcolor = 1;

if(flip==0){
  if(s >= t){
    if(fore_direction == 1)
    surfcolor =  mix(forecolor_a, forecolor_b, s);
    else
    surfcolor =  mix(forecolor_a, forecolor_b, t);
  }
  else{
    if(back_direction == 1)
    surfcolor = mix(backcolor_a, backcolor_b, s );
    else
    surfcolor = mix(backcolor_a, backcolor_b, t );
  }
}

else{
  if(s <= -t+1){
    if(fore_direction == 1)
    surfcolor =  mix(forecolor_a, forecolor_b, s);
    else
    surfcolor =  mix(forecolor_a, forecolor_b, t);
  }
  else{
    if(back_direction == 1)
    surfcolor = mix(backcolor_a, backcolor_b, s );
    else
    surfcolor = mix(backcolor_a, backcolor_b, t );
  }
}


/* STEP 1 - set the apparent surface opacity */
Oi = Os;
```

```
/* STEP 2 - calculate the apparent surface color */
Ci = Oi * Cs * surfcolor * Kfb;
}
```

```
/* shader_src/cross.sl */
surface
cross(  float  Kfb = 1;
     float rec1_t_center = 0.5;
     float rec1_height = 0.3;
     float rec2_s_center = 0.5;
     float rec2_width = 0.3;
     color backcolor_a = color(0,1,0);
     color backcolor_b = color(0,1,1);
     color forecolor_a = color(1,1,0);
     color forecolor_b = color(1,0,0);
     float fore_direction = 1 /* [0 or 1] */;
     float back_direction = 1 /* [0 or 1] */;
  )
{
color  surfcolor = 1;
float rec1_s_center = 0.5;
float rec2_t_center = 0.5;
float rec1_width = 1;
float rec2_height = 1;
float    rec1_h = rec1_height/2;
float    rec1_w = rec1_width/2;
float    rec2_h = rec2_height/2;
float    rec2_w = rec2_width/2;
if((s >= rec1_s_center - rec1_w && s <= rec1_s_center + rec1_w
  &&
  t >= rec1_t_center - rec1_h && t <= rec1_t_center + rec1_h)
  ||
  (s >= rec2_s_center - rec2_w && s <= rec2_s_center + rec2_w
  &&
  t >= rec2_t_center - rec2_h && t <= rec2_t_center + rec2_h)){

    if(fore_direction == 1)
    surfcolor =  mix(forecolor_a, forecolor_b, s);
    else
    surfcolor =  mix(forecolor_a, forecolor_b, t);

  }

  else{
    if(back_direction == 1)
    surfcolor = mix(backcolor_a, backcolor_b, s );
    else
    surfcolor = mix(backcolor_a, backcolor_b, t );
  }
```

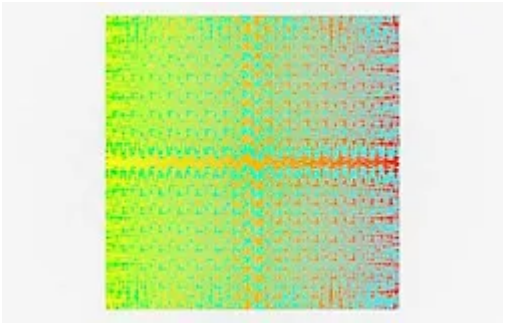
```
/* STEP 1 - set the apparent surface opacity */
Oi = Os;

/* STEP 2 - calculate the apparent surface color */
Ci = Oi * Cs * surfcolor * Kfb;
}
```

```
/* shader_src/circle.sl */
surface
circle(  float  Kfb = 1;
   float s_center = 0.7;
   float t_center = 0.5;
   float radius = 0.3;
   color backcolor_a = color(0,1,0);
   color backcolor_b = color(0,1,1);
   color forecolor_a = color(1,1,0);
   color forecolor_b = color(1,0,0);
   float fore_direction = 1 /* [0 or 1] */;
   float back_direction = 1 /* [0 or 1] */;

   )


{
color  surfcolor = 1;
float d=distance (point(s,t,0),point(s_center,t_center,0));

   if(d <= radius){
    if(fore_direction == 1)
    surfcolor =  mix(forecolor_a, forecolor_b, s);
    else
    surfcolor =  mix(forecolor_a, forecolor_b, t);
    }

   else{
    if(back_direction == 1)
    surfcolor = mix(backcolor_a, backcolor_b, s );
    else
    surfcolor = mix(backcolor_a, backcolor_b, t );
   }


Oi = Os;

/* STEP 2 - calculate the apparent surface color */
Ci = Oi * Cs * surfcolor * Kfb;
}
```

```
/* shader_src/rectangle.sl */
surface
rectangle(  float  Kfb = 1;
     float s_center = 0.5;
     float t_center = 0.5;
     float width = 0.4;
     float height = 0.3;
     color backcolor_a = color(0,1,0);
     color backcolor_b = color(0,1,1);
     color forecolor_a = color(1,1,0);
     color forecolor_b = color(1,0,0);
     float fore_direction = 1 /* [0 or 1] */;
     float back_direction = 1 /* [0 or 1] */;
   )
{
color  surfcolor =1;
float    h = height/2;
float    w = width/2;

   if(s >= s_center - w && s <= s_center + w
     &&
     t >= t_center - h && t <= t_center + h){
     if(fore_direction == 1)
     surfcolor =  mix(forecolor_a, forecolor_b, s);
     else
     surfcolor =  mix(forecolor_a, forecolor_b, t);
     }

   else{
     if(back_direction == 1)
     surfcolor = mix(backcolor_a, backcolor_b, s );
     else
     surfcolor = mix(backcolor_a, backcolor_b, t );
   }


/* STEP 1 - set the apparent surface opacity */
Oi = Os;

/* STEP 2 - calculate the apparent surface color */
Ci = Oi * Cs * surfcolor * Kfb;
}
```

```
/* shader_src/heart.sl */
surface
heart(  float  Kfb = 1;
   float size = 0.33;
   float s_offset = 0.5;
   float t_offset = 0.5;
   color backcolor_a = color(0,1,0);
   color backcolor_b = color(0,1,1);
   color forecolor_a = color(1,1,0);
   color forecolor_b = color(1,0,0);
   float fore_direction = 1 /* [0 or 1] */;
   float back_direction = 1 /* [0 or 1] */;

   )



{

color  surfcolor = 1;
float a = s-s_offset;
float b = t-t_offset;
float c = 1/size;
float x = c*a;
float y = c*b;

   if((x*x+y*y-1)*(x*x+y*y-1)*(x*x+y*y-1)-x*x*y*y*y<=0){
     if(fore_direction == 1)
     surfcolor =  mix(forecolor_a, forecolor_b, s);
     else
     surfcolor =  mix(forecolor_a, forecolor_b, t);
     }

   else{
     if(back_direction == 1)
     surfcolor = mix(backcolor_a, backcolor_b, s );
     else
     surfcolor = mix(backcolor_a, backcolor_b, t );
   }

Oi = Os;

/* STEP 2 - calculate the apparent surface color */
Ci = Oi * Cs * surfcolor * Kfb;
}
```

```
/* shader_src/wave.sl */
surface
wave(  float  Kfb = 1;
   float s_offset = 0.5;
   float t_offset = 0.5;
   float size = 0.01;
   color backcolor_a = color(0,1,0);
   color backcolor_b = color(0,1,1);
   color forecolor_a = color(1,1,0);
   color forecolor_b = color(1,0,0);
   float fore_direction = 1 /* [0 or 1] */;
   float back_direction = 1 /* [0 or 1] */;

   )


{
color  surfcolor = 1;
float a = s-s_offset;
float b = t-t_offset;
float c = 1/size;
float x = c*a;
float y = c*b;



   if( x <= 11*cos(y) - 6*cos(y*11/6) ){
     if(fore_direction == 1)
     surfcolor =  mix(forecolor_a, forecolor_b, s);
     else
     surfcolor =  mix(forecolor_a, forecolor_b, t);
     }

   else{
     if(back_direction == 1)
     surfcolor = mix(backcolor_a, backcolor_b, s );
     else
     surfcolor = mix(backcolor_a, backcolor_b, t );
   }


Oi = Os;

/* STEP 2 - calculate the apparent surface color */
Ci = Oi * Cs * surfcolor * Kfb;
}
```

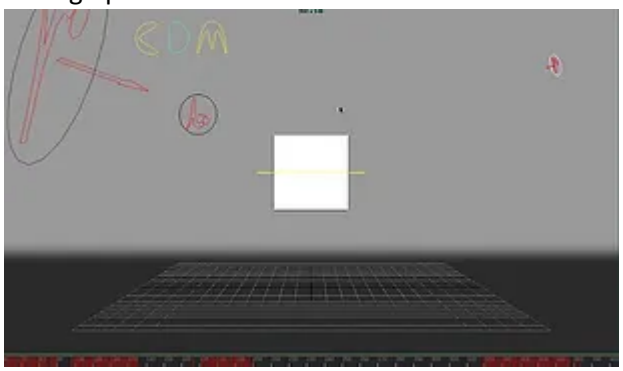


```
/* shader_src/weird.sl */
surface
weird(  float  Kfb = 1;
   float s_offset = 0.5;
   float t_offset = 0.5;
   float size = 0.01;
   color backcolor_a = color(0,1,0);
   color backcolor_b = color(0,1,1);
   color forecolor_a = color(1,1,0);
   color forecolor_b = color(1,0,0);
   float fore_direction = 1 /* [0 or 1] */;
   float back_direction = 1 /* [0 or 1] */;

   )


{
color  surfcolor = 1;
float a = s-s_offset;
float b = t-t_offset;
float c = 1/size;
float x = c*a;
float y = c*b;


   if(
     sin(x* sin(y)) <= cos(y*cos(x))

   ){
     if(fore_direction == 1)
     surfcolor =  mix(forecolor_a, forecolor_b, s);
     else
     surfcolor =  mix(forecolor_a, forecolor_b, t);
     }

   else{
     if(back_direction == 1)
     surfcolor = mix(backcolor_a, backcolor_b, s );
     else
     surfcolor = mix(backcolor_a, backcolor_b, t );
   }


Oi = Os;

/* STEP 2 - calculate the apparent surface color */
Ci = Oi * Cs * surfcolor * Kfb;
}
```

Make use of the above custom nodes:

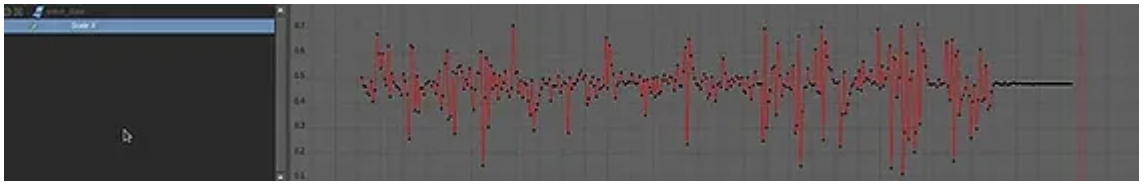
The notes on this page explain how custom Slim nodes written using the RenderMan Shading Language can be combined with sophisticated physically plausible nodes that ship with Pixar's RenderMan Studio.

Setting up the scene and controls:



It started off with three-point lighting and a simple cube. Then I created four controls: Main, Color, Displacement and Mask, in which attributes were made to connect to shape attributes that were connected to the slim node via TCL expression. Animating under objects instead of shapes is a lot more convenient.

Animating with soundwave node:

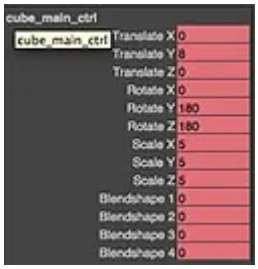


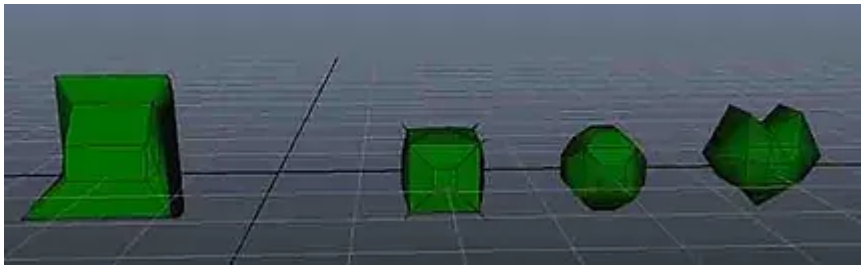
1. load audiowave plugin, create an audiowave node (bonus tool) and link it to the sound file
2. create a nurbsSphere
3. run mel srcript:

```
connectAttr -f audioWave1.output nurbsSphere1.scaleX;

connectAttr -f time1.outTime audioWave1.input;
```

4. bake the nurbsSphere keys
5. then I locked the Sphere scaleX attribute and whenever I need the keys data, I can copy it from the graph editor

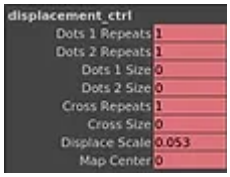
Main Control and blendshapes:

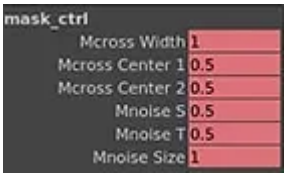




The main control is responsible for translate, rotate, scale and the four blend shapes (connected to Shape node blend shape attributes).

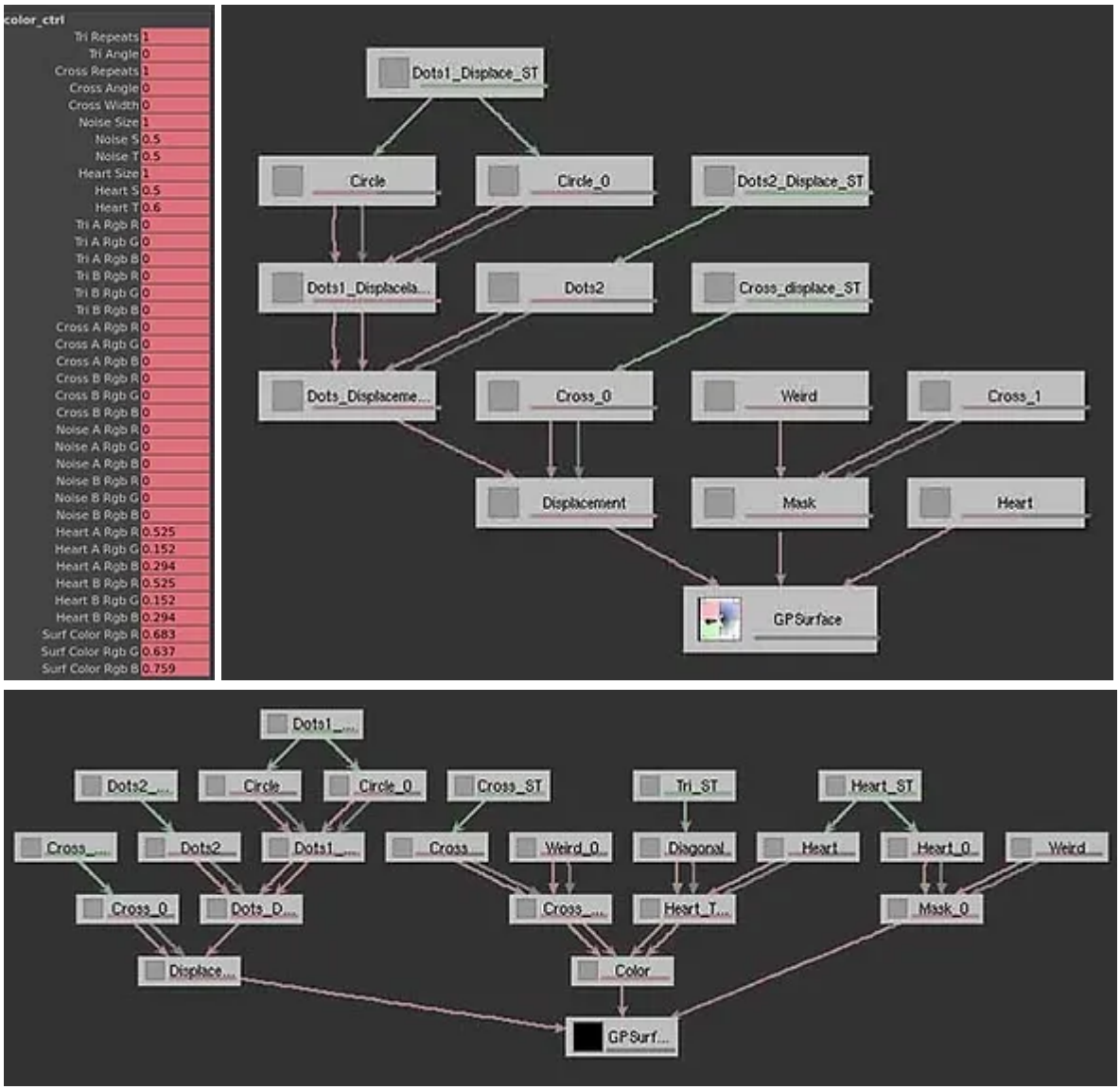
Displacement and Mask control:





The displacement and mask control are responsible for the displacement scale, repetitions (ST), sizes of the shapes and etc that are connected as displacement or mask for the GPSurface.

Color control:



Three attributes are connected to the R, G and B values for each color. With all the different shapes and mixed colors (see pic1), the list of attributes in the color control was long (see pic2). But then I realized all these variations in colors and patterns actually looked bad, so I renewed the slim node, made it less complicated and then deleted all the unnecessary attributes.

Progress 1: https://youtu.be/Fab67bxlRE4

I realized that the motion is much more interesting with displacement

Progress 2: https://youtu.be/ZRMZBNncjoU

I added colors to the lights, fixed the second half as planned and animated colors and camera movements, but the timing in the second half was off; the camera movements seemed weird and the random colors were sometimes distracting.