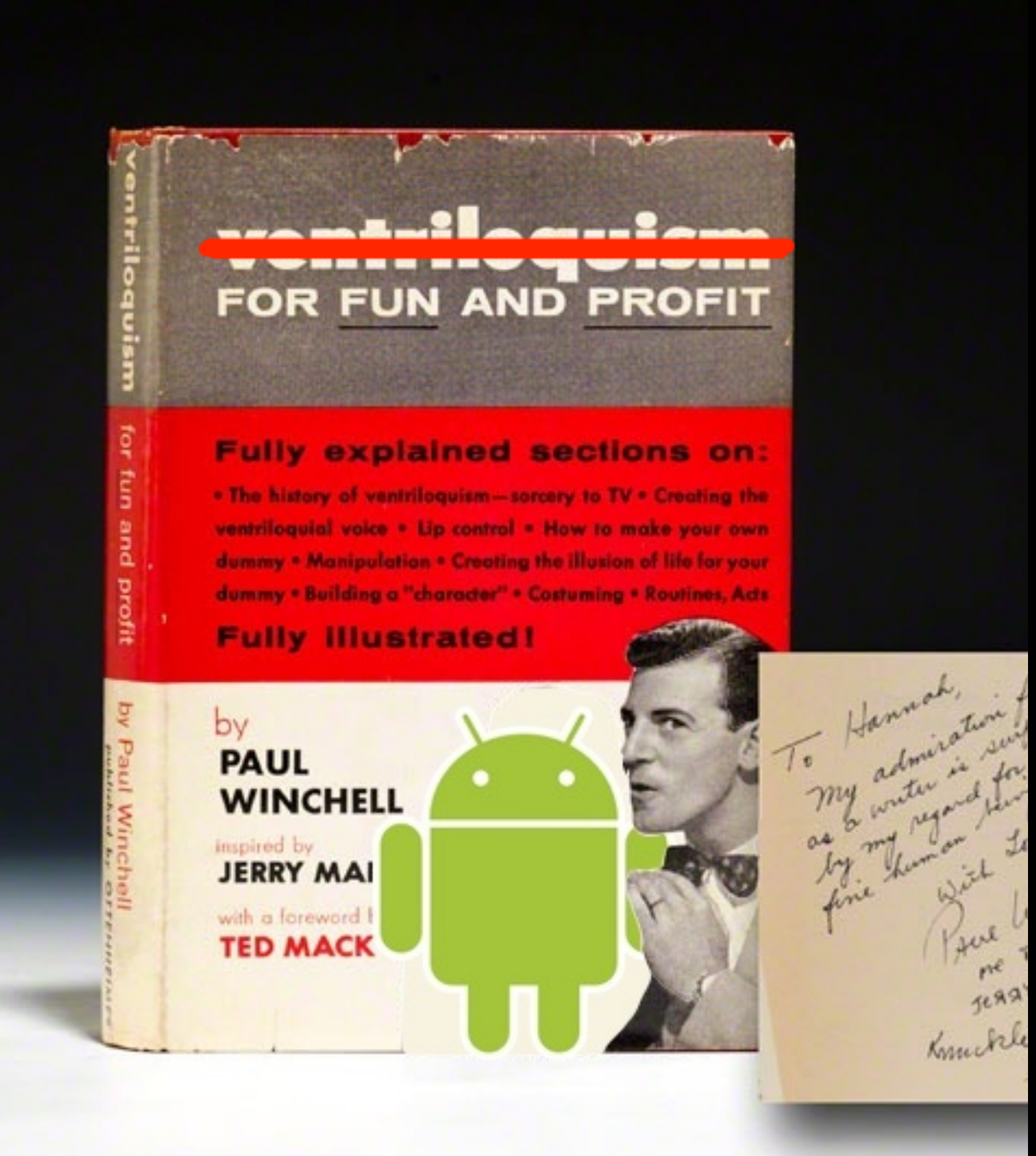
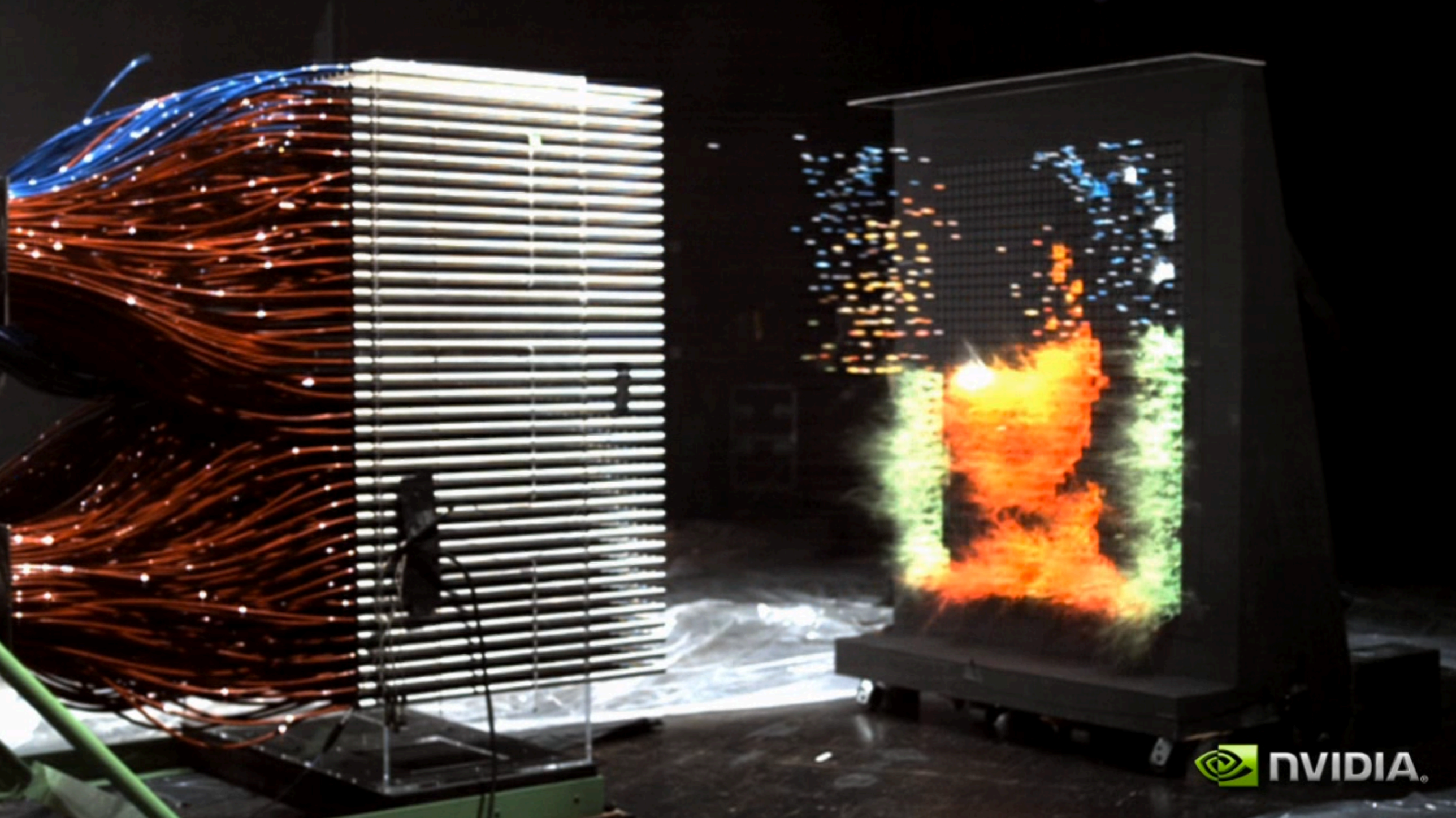
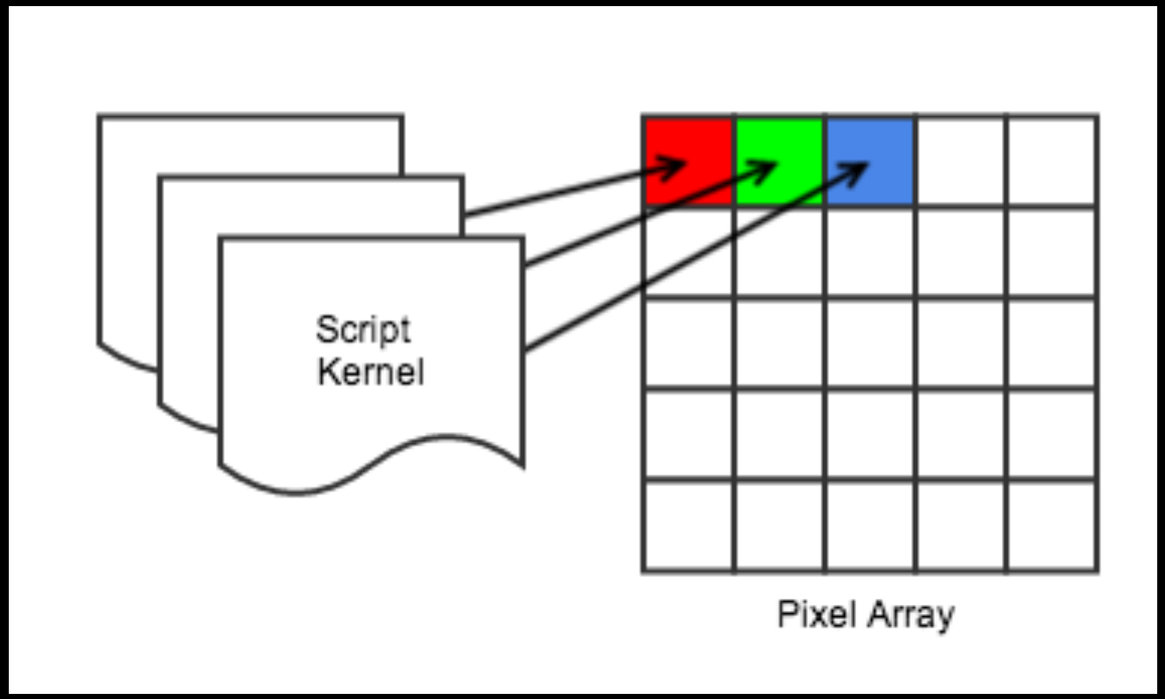


RENDERSCRIPT
FOR FUN AND PROFIT

DATA-PARALLEL COMPUTATION...
WHATNOW??


NVIDIA

PARALLEL = SPEED



Why Choose?

DSP Cores

+

GPU Cores

UGH, NOW I HAVE TO LEARN C, RIGHT?

ScriptIntrinsic3DLUT

ScriptIntrinsicBlur

ScriptIntrinsicHistogram

ScriptIntrinsicLUT

ScriptIntrinsicYuvToRGB

9 Types of ScripIntrinsics

```
private Bitmap blurBitmap(Bitmap sourceBitmap) {
    // First, we need a renderscript context.
    RenderScript rs = RenderScript.create(MainActivity.this);

    // We create an empty bitmap matching the source bitmap size and configuration.
    Bitmap outputBitmap = Bitmap.createBitmap(
            sourceBitmap.getWidth(),
            sourceBitmap.getHeight(),
            sourceBitmap.getConfig());

    // Allocations are the primary method through which we pass data to and from a Renderscript 'kernel'.
    // (More on this later)
    Allocation tmpIn = Allocation.createFromBitmap(rs, sourceBitmap);
    Allocation tmpOut = Allocation.createFromBitmap(rs, outputBitmap);


    // We have a pick of 9 different renderscript "intrinsics", here we pick a blur.
    ScriptIntrinsicBlur theIntrinsic = ScriptIntrinsicBlur.create(rs, Element.U8_4(rs));

    // Sets the desired blur radius.
    theIntrinsic.setRadius(24.f);
    // Sets the input of the blur.
    theIntrinsic.setInput(tmpIn);
    // Applies the filter to the input, and saves it to the tmpOut Allocation.
    theIntrinsic.forEach(tmpOut);


    // Small safety measure against memory issues that can creep up when manipulating bitmaps.
    sourceBitmap.recycle();

    // Copy from allocation into the output bitmap.
    tmpOut.copyTo(outputBitmap);

    // And we're done!
    rs.destroy();

    return outputBitmap;
}
```

RenderScript context

Intrinsic

Output Bitmap

Allocations

Applies Intrinsic

```
private Bitmap blurBitmap(Bitmap sourceBitmap) {
    // First, we need a renderscript context.
    RenderScript rs = RenderScript.create(MainActivity.this);

    // We create an empty bitmap matching the source bitmap size and configuration.
    Bitmap outputBitmap = Bitmap.createBitmap(
            sourceBitmap.getWidth(),
            sourceBitmap.getHeight(),
            sourceBitmap.getConfig());

    // Allocations are the primary method through which we pass data to and from a Renderscript 'kernel'.
    // (More on this later)
    Allocation tmpIn = Allocation.createFromBitmap(rs, sourceBitmap);
    Allocation tmpOut = Allocation.createFromBitmap(rs, outputBitmap);


    // We have a pick of 9 different renderscript "intrinsics", here we pick a blur.
    ScriptIntrinsicBlur theIntrinsic = ScriptIntrinsicBlur.create(rs, Element.U8_4(rs));

    // Sets the desired blur radius.
    theIntrinsic.setRadius(24.f);
    // Sets the input of the blur.
    theIntrinsic.setInput(tmpIn);
    // Applies the filter to the input, and saves it to the tmpOut Allocation.
    theIntrinsic.forEach(tmpOut);


    // Small safety measure against memory issues that can creep up when manipulating bitmaps.
    sourceBitmap.recycle();

    // Copy from allocation into the output bitmap.
    tmpOut.copyTo(outputBitmap);

    // And we're done!
    rs.destroy();

    return outputBitmap;
}
```

Renderscript Ignition

Super Efficient Bitmap Blurring!

TODO!

GitHub Project

C99 LANGUAGE [.RS FILES]

<project>/src/main/rs/*.rs

```
void init_vignette(uint32_t dim_x, uint32_t dim_y, float center_x, float center_y,
        float desired_scale, float desired_shade, float desired_slope) {

    neg_center.x = -center_x;
    neg_center.y = -center_y;
    inv_dimensions.x = 1.f / (float)dim_x;
    inv_dimensions.y = 1.f / (float)dim_y;

    axis_scale = (float2)1.f;
    if (dim_x > dim_y)
        axis_scale.y = (float)dim_y / (float)dim_x;
    else
        axis_scale.x = (float)dim_x / (float)dim_y;

    const float max_dist = 0.5 * length(axis_scale);
    sloped_inv_max_dist = desired_slope * 1.f/max_dist;

    // Range needs to be between 1.3 to 0.6. When scale is zero then range is
    // 1.3 which means no vignette at all because the luminousity difference is
    // less than 1/256.  Expect input scale to be between 0.0 and 1.0.
    const float neg_range = 0.7*sqrt(desired_scale) - 1.3;
    sloped_neg_range = exp(neg_range * desired_slope);

    shade = desired_shade;
    opp_shade = 1.f - desired_shade;
}

void root(const uchar4 *in, uchar4 *out, uint32_t x, uint32_t y) {
    // Convert x and y to floating point coordinates with center as origin
    const float4 fin = convert_float4(*in);
    const float2 inCoord = {(float)x, (float)y};
    const float2 coord = mad(inCoord, inv_dimensions, neg_center);
    const float sloped_dist_ratio = length(axis_scale * coord)  * sloped_inv_max_dist;
    const float lumen = opp_shade + shade / ( 1.0 + sloped_neg_range * exp(sloped_dist_ratio) );
    float4 fout;
    fout.rgb = fin.rgb * lumen;
    fout.w = fin.w;
    *out = convert_uchar4(fout);
}
```

```
void init_vignette(uint32_t dim_x, uint32_t dim_y, float center_x, float center_y,
        float desired_scale, float desired_shade, float desired_slope) {

    neg_center.x = -center_x;
    neg_center.y = -center_y;
    inv_dimensions.x = 1.f / (float)dim_x;
    inv_dimensions.y = 1.f / (float)dim_y;

    axis_scale = (float2)1.f;
    if (dim_x > dim_y)
        axis_scale.y = (float)dim_y / (float)dim_x;
    else
        axis_scale.x = (float)dim_x / (float)dim_y;

    const float max_dist = 0.5 * length(axis_scale);
    sloped_inv_max_dist = desired_slope * 1.f/max_dist;

    // Range needs to be between 1.3 to 0.6. When scale is zero then range is
    // 1.3 which means no vignette at all because the luminousity difference is
    // less than 1/256.  Expect input scale to be between 0.0 and 1.0.
    const float neg_range = 0.7*sqrt(desired_scale) - 1.3;
    sloped_neg_range = exp(neg_range * desired_slope);

    shade = desired_shade;
    opp_shade = 1.f - shade;
}

void root(const uchar4 *in, uchar4 *out, uint32_t x, uint32_t y) {
    // Convert x and y to floating point coordinates with center as origin
    const float4 fin = convert_float4(*in);
    const float2 inCoord = {(float)x, (float)y};
    const float2 coord = mad(inCoord, inv_dimensions, neg_center);
    const float sloped_dist_ratio = length(axis_scale * coord)  * sloped_inv_max_dist;
    const float lumen = opp_shade + shade / ( 1.0 + sloped_neg_range * exp(sloped_dist_ratio) );
    float4 fout;
    fout.rgb = fin.rgb * lumen;
    fout.w = fin.w;
    *out = convert_uchar4(fout);
}
```

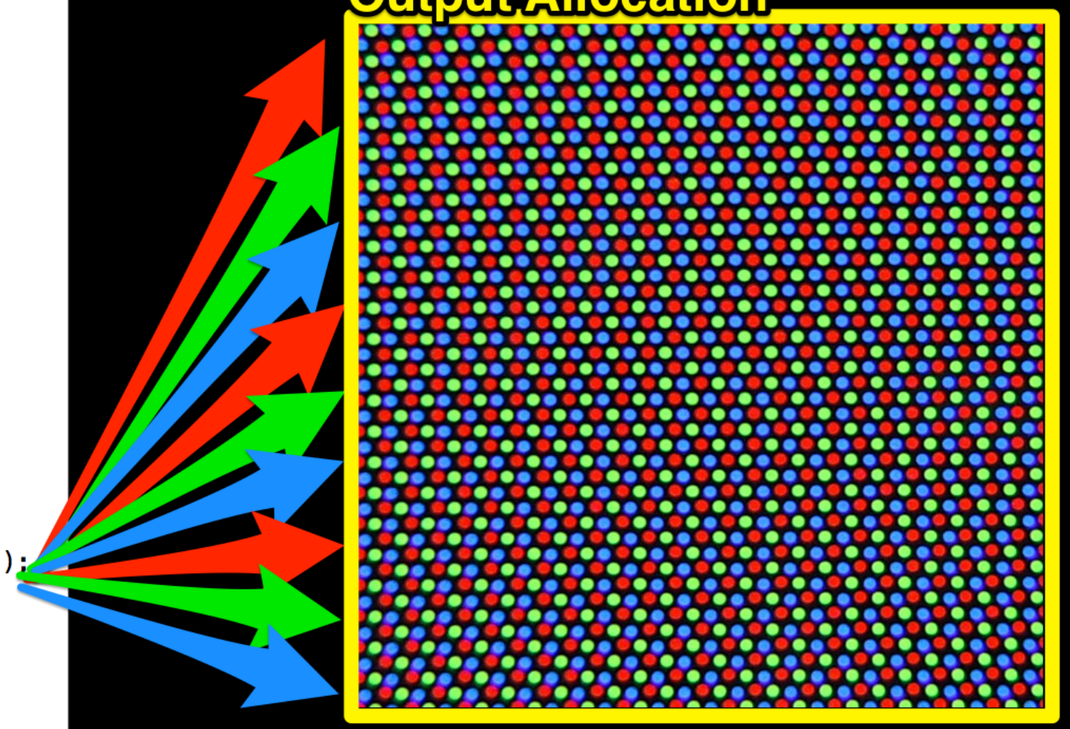
init function

Input Element

kernel function

Output Element

Output Allocation



```
private Bitmap vignetteBitmap(Bitmap sourceBitmap) {
    // First, we need a renderscript context.
    RenderScript rs = RenderScript.create(MainActivity.this);

    // We create an empty bitmap matching the source bitmap size and configuration.
    Bitmap outputBitmap = Bitmap.createBitmap(sourceBitmap.getWidth(), sourceBitmap.getHeight(),

    // Allocations are the primary method through which we pass data to and from a Renderscript 'kernel'.
    // (More on this later)
    Allocation tmpIn = Allocation.createFromBitmap(rs, sourceBitmap);
    Allocation tmpOut = Allocation.createFromBitmap(rs, outputBitmap);

    // This instantiates the generated script. Naming convention is ScriptC_xyz
    ScriptC_vignette_full scriptVignette = new ScriptC_vignette_full(rs,getResources(),R.raw.vignette_full);

    // Vignette parameter assignments
    float center_x = 1f; float center_y = 0.5f;
    float scale = 0.8f;
    float shade = 0.8f; // 0.0f light -> 1.0f dark
    float slope = 5.0f; // Rate at which we swing from dark to light

    // Invoke the vignette script with parameters.
    scriptVignette.invoke_init_vignette(
            tmpIn.getType().getX(), tmpIn.getType().getY(),
            center_x, center_y, scale, shade, slope);

    // Execute
    scriptVignette.forEach_root(tmpIn, tmpOut);


    // Small safety measure against memory issues that can creep up when manipulating bitmaps.
    sourceBitmap.recycle();

    // Copy from allocation into the output bitmap.
    tmpOut.copyTo(outputBitmap);

    // And we're done!
    rs.destroy();

    return outputBitmap;
}
```

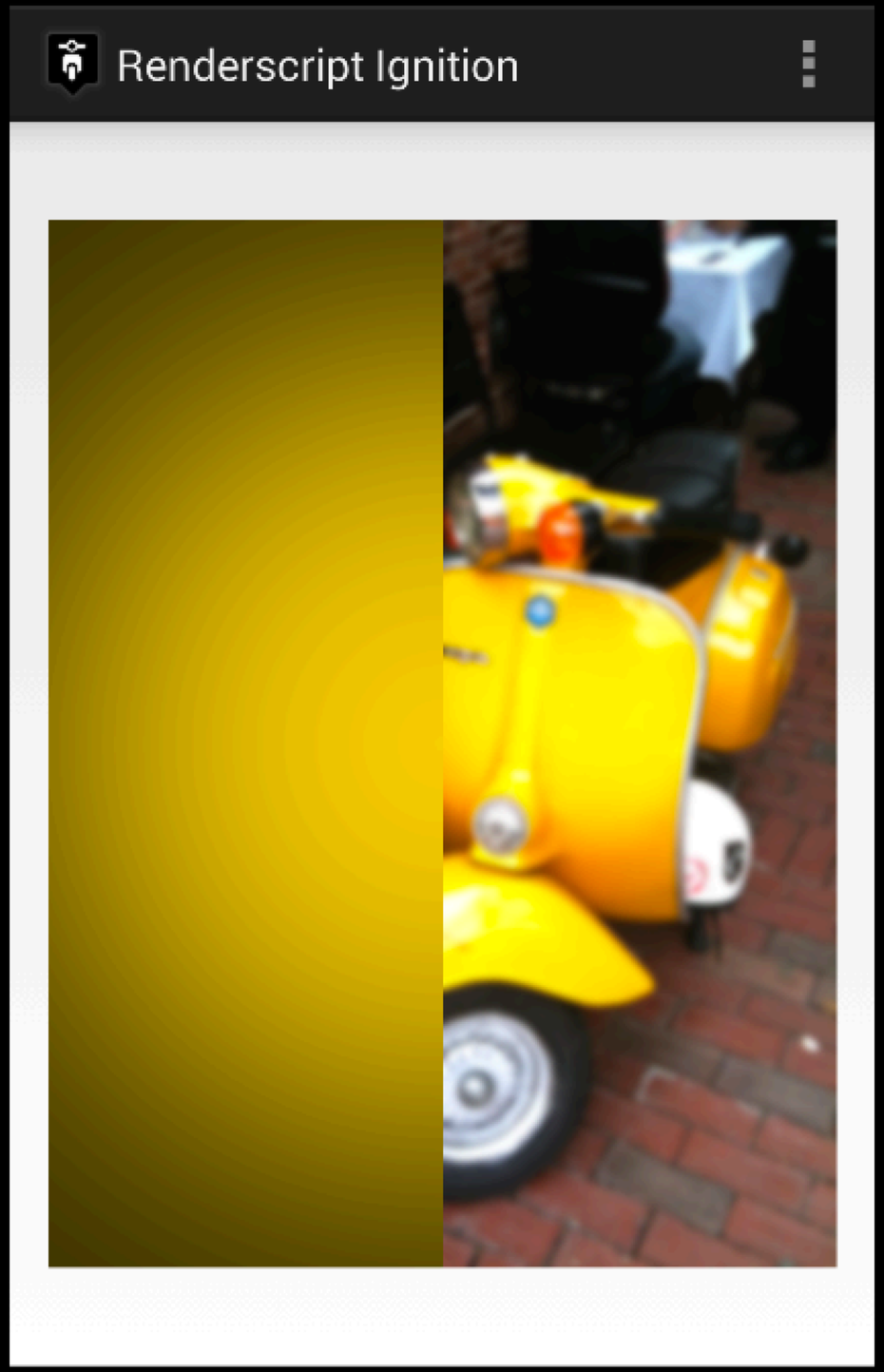
TODO:

Welcome Back!

maps to .rs

maps to init function

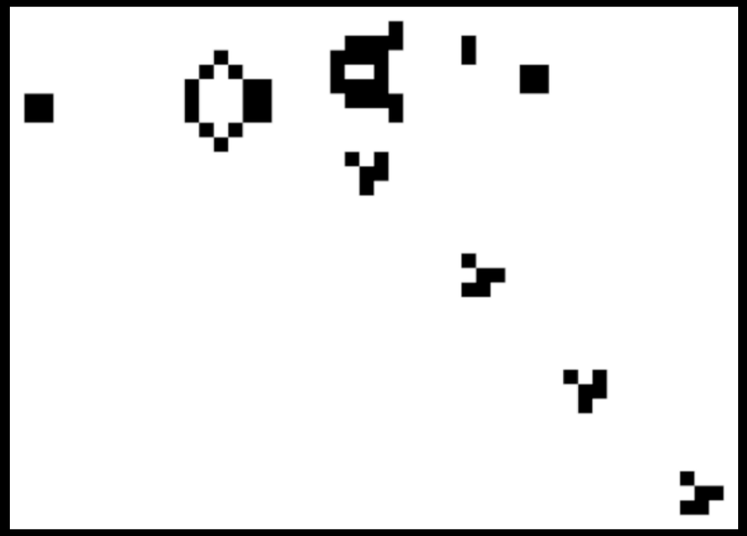
maps to kernel function



VIGNETTE

[on yellow background for emphasis]

GAME OF LIFE



1. Neighbours < 2

   » Death

2. Neighbours 2 | 3

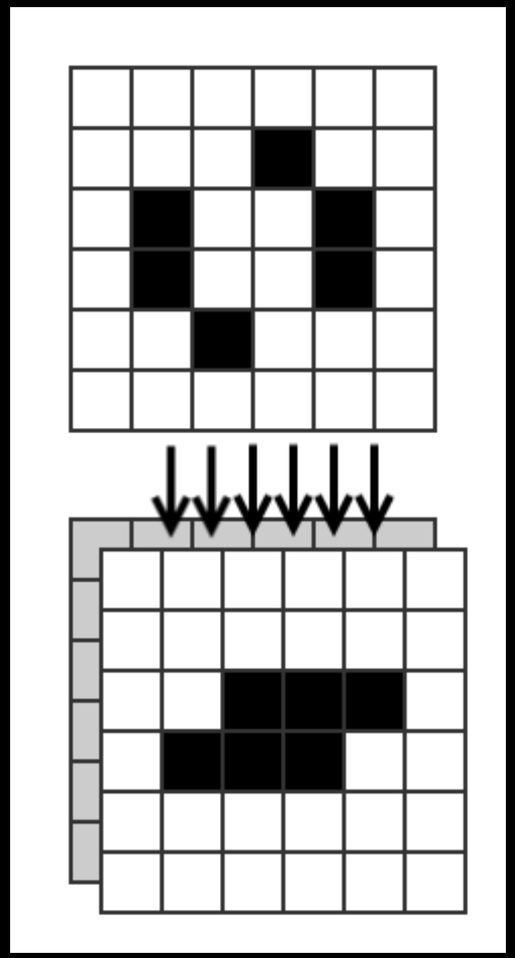
   » Lives

3. Neighbours > 3

   » Death

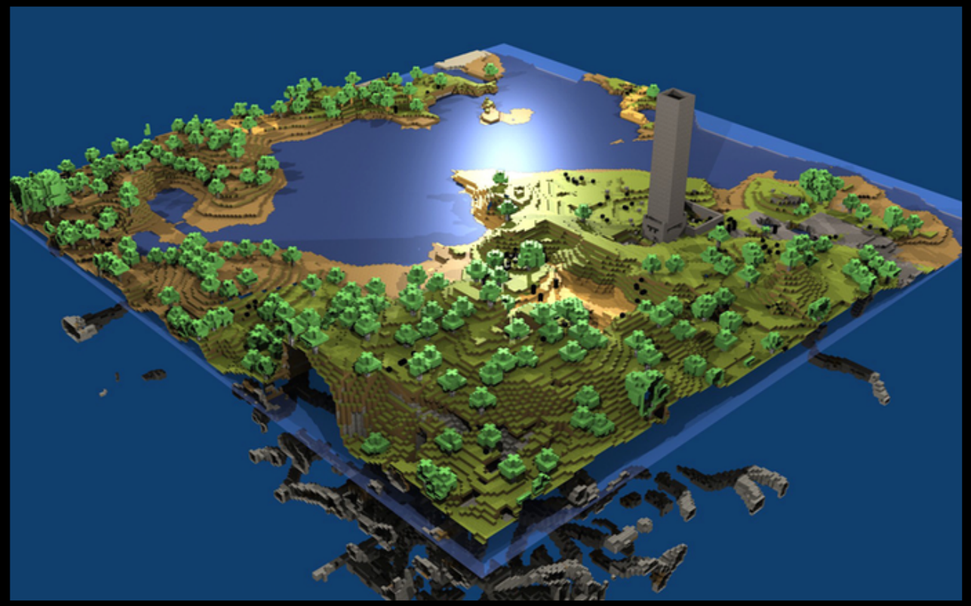
4. Neighbours == 3

   » Birth



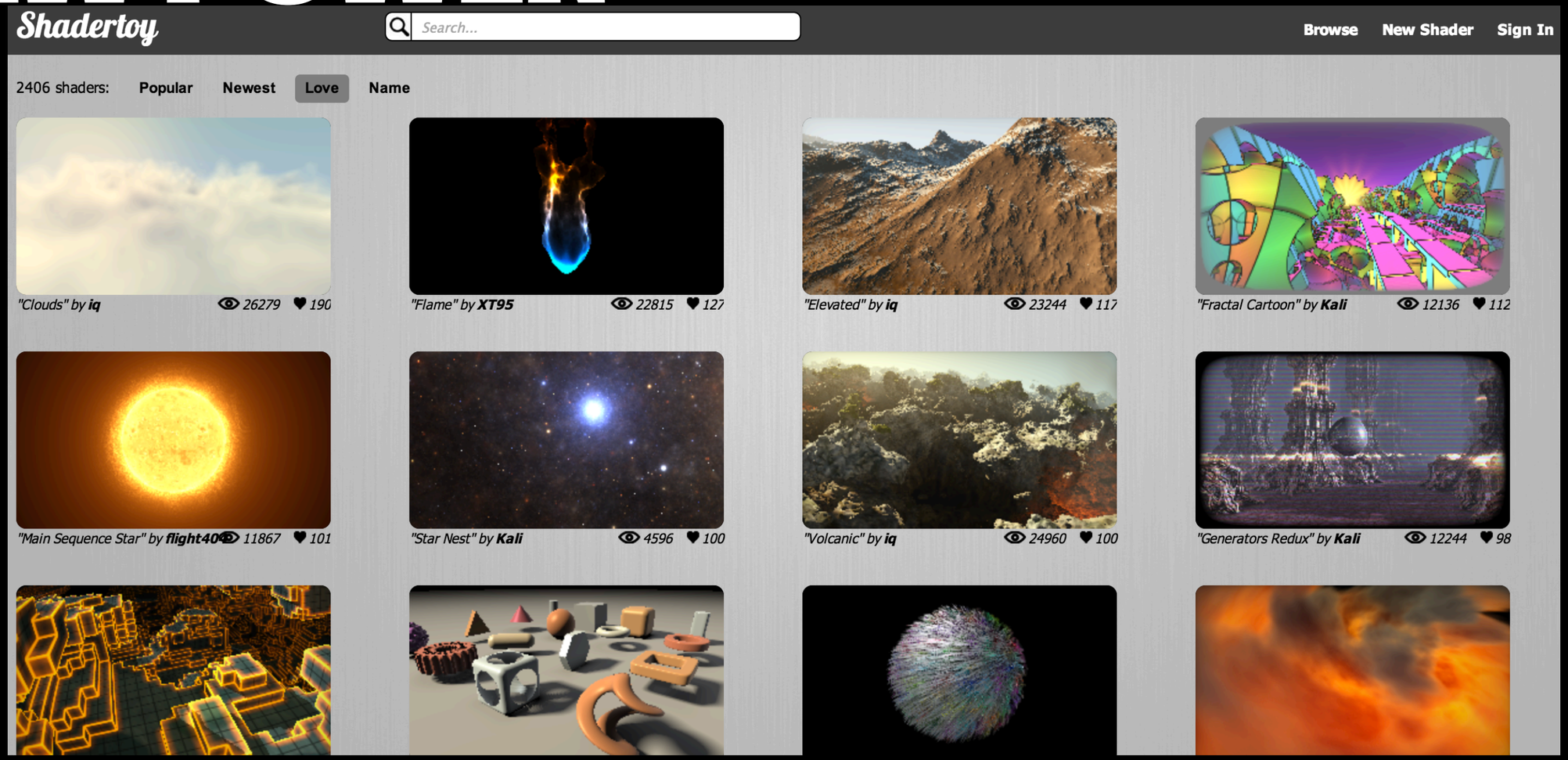
PARALLEL COMPUTATION

We depend only on current state of system, not neighboring cell's computation result.

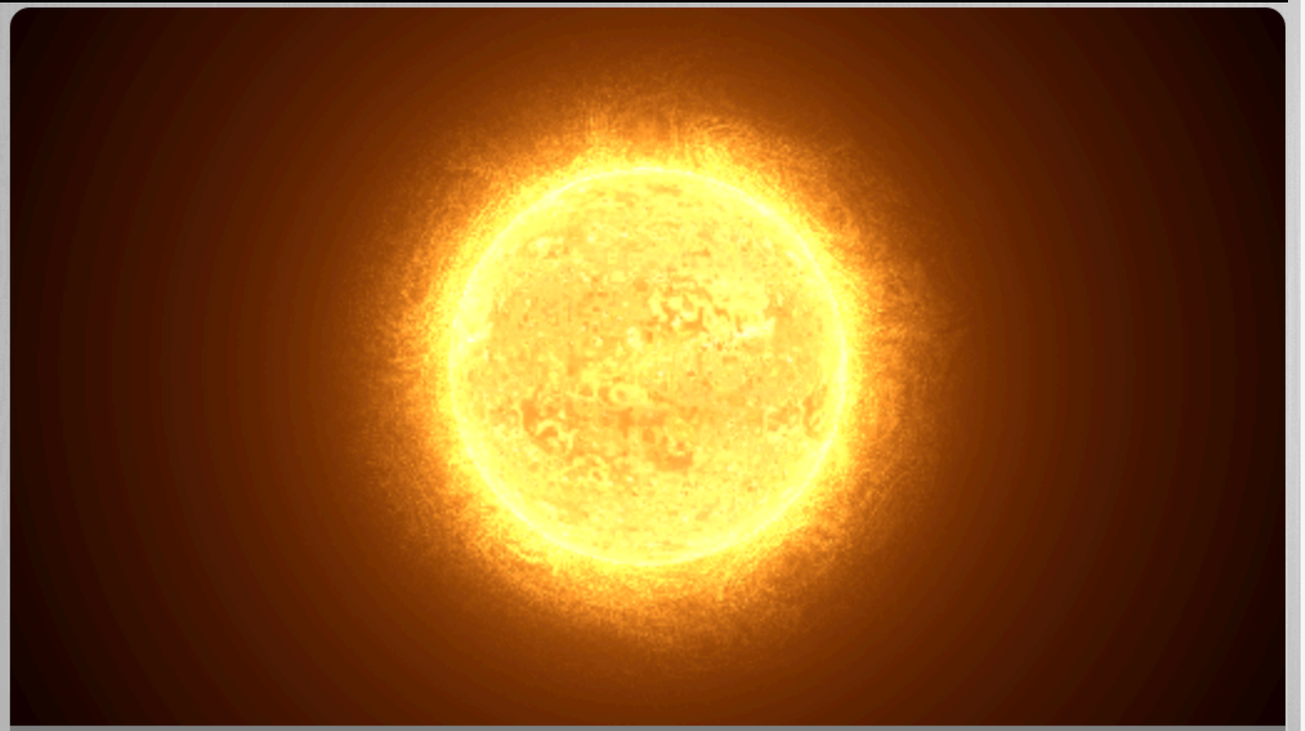



METAL MANIA

RAW POWER


Shadertoy
Search...
Browse
New Shader
Sign In
2406 shaders:
Popular
Newest
Love
Name
"Clouds" by iq 26279 190
"Flame" by XT95 22815 127
"Elevated" by iq 23244 117
"Fractal Cartoon" by Kali 12136 112
"Main Sequence Star" by flight40 11867 101
"Star Nest" by Kali 4596 100
"Volcanic" by iq 24960 100
"Generators Redux" by Kali 12244 98

SMALL FOOTPRINT




```
56
57      float newTime1  = abs( snoise( coord + vec3( 0.0, -time * ( 0.35 + brightness * 0.001 ), time * 0.015 ), 15
58      float newTime2  = abs( snoise( coord + vec3( 0.0, -time * ( 0.15 + brightness * 0.001 ), time * 0.015 ), 45
59      for( int i=1; i<=7; i++ ){
60          float power = pow( 2.0, float(i + 1) );
61          fVal1 += ( 0.5 / power ) * snoise( coord + vec3( 0.0, -time, time * 0.2 ), ( power * ( 10.0 ) * ( newTi
62          fVal2 += ( 0.5 / power ) * snoise( coord + vec3( 0.0, -time, time * 0.2 ), ( power * ( 25.0 ) * ( newTi
63      }
64
65      float corona        = pow( fVal1 * max( 1.1 - fade, 0.0 ), 2.0 ) * 50.0;
66      corona              += pow( fVal2 * max( 1.1 - fade, 0.0 ), 2.0 ) * 50.0;
67      corona              *= 1.2 - newTime1;
68      vec3 sphereNormal   = vec3( 0.0, 0.0, 1.0 );
69      vec3 dir            = vec3( 0.0 );
70      vec3 center         = vec3( 0.5, 0.5, 1.0 );
71      vec3 starSphere     = vec3( 0.0 );
72
73      vec2 sp = -1.0 + 2.0 * uv;
74      sp.x *= aspect;
75      sp *= ( 2.0 - brightness );
76      float r = dot(sp,sp);
77      float f = (1.0-sqrt(abs(1.0-r)))/(r) + brightness * 0.5;
78      if( dist < radius ){
79          corona          *= pow( dist * invRadius, 24.0 );
80          vec2 newUv;
81          newUv.x = sp.x*f;
82          newUv.y = sp.y*f;
83          newUv += vec2( time, 0.0 );
84
85          vec3 texSample  = texture2D( iChannel0, newUv ).rgb;
86          float uOff      = ( texSample.g * brightness * 4.5 + time );
87          vec2 starUV     = newUv + vec2( uOff, 0.0 );
88          starSphere      = texture2D( iChannel0, starUV ).rgb;
89      }
90
91      float starGlow  = min( max( 1.0 - dist * ( 1.0 - brightness ), 0.0 ), 1.0 );
92      //gl_FragColor.rgb  = vec3( r );
93      gl_FragColor.rgb    = vec3( f * ( 0.75 + brightness * 0.3 ) * orange ) + starSphere + corona * orange + sta
94      gl_FragColor.a      = 1.0;
95  }
96
97
```

2329 chars

THANK YOU!