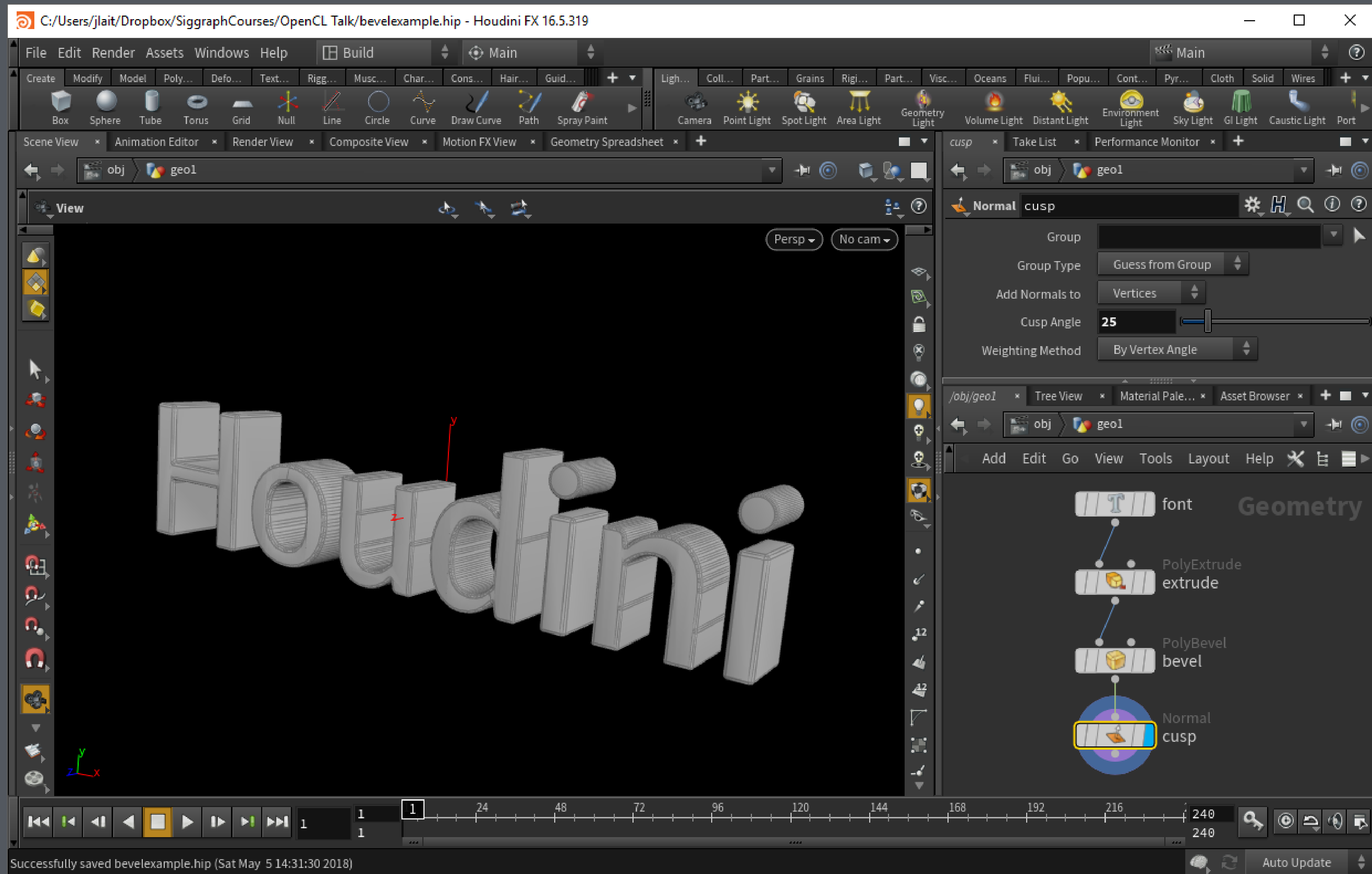


Exposing Artists to OpenCL

Jeff Lait | SideFX

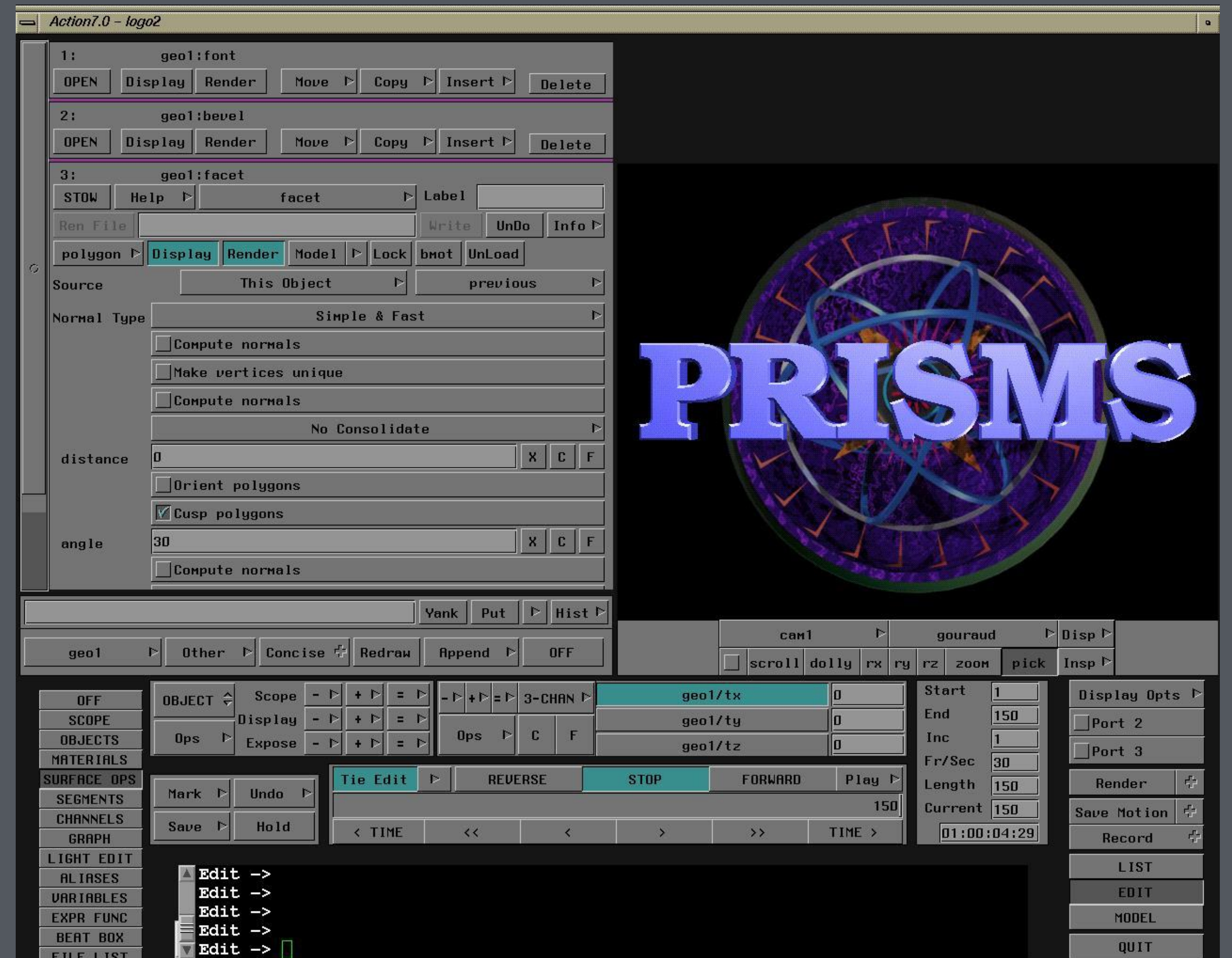


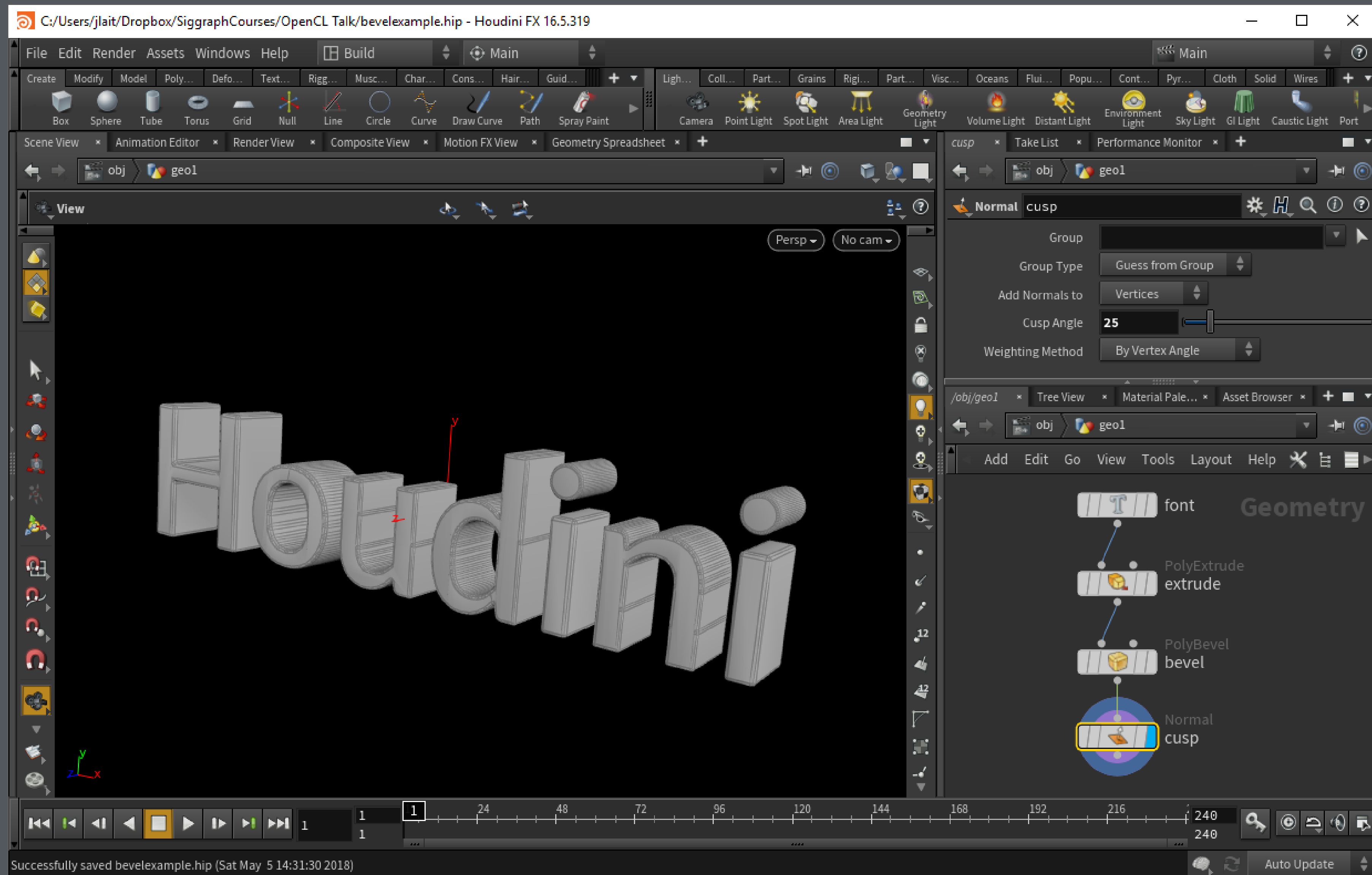


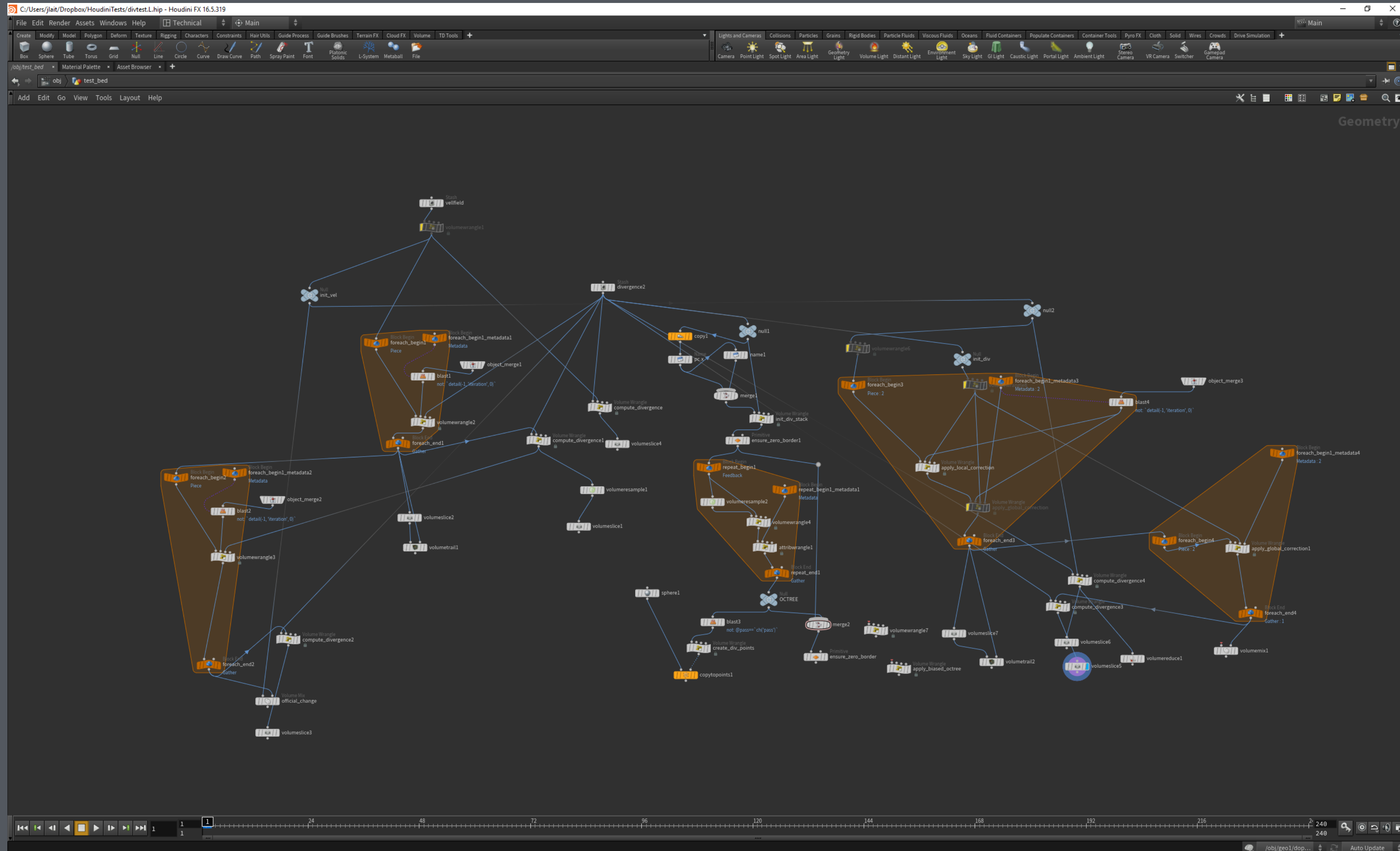


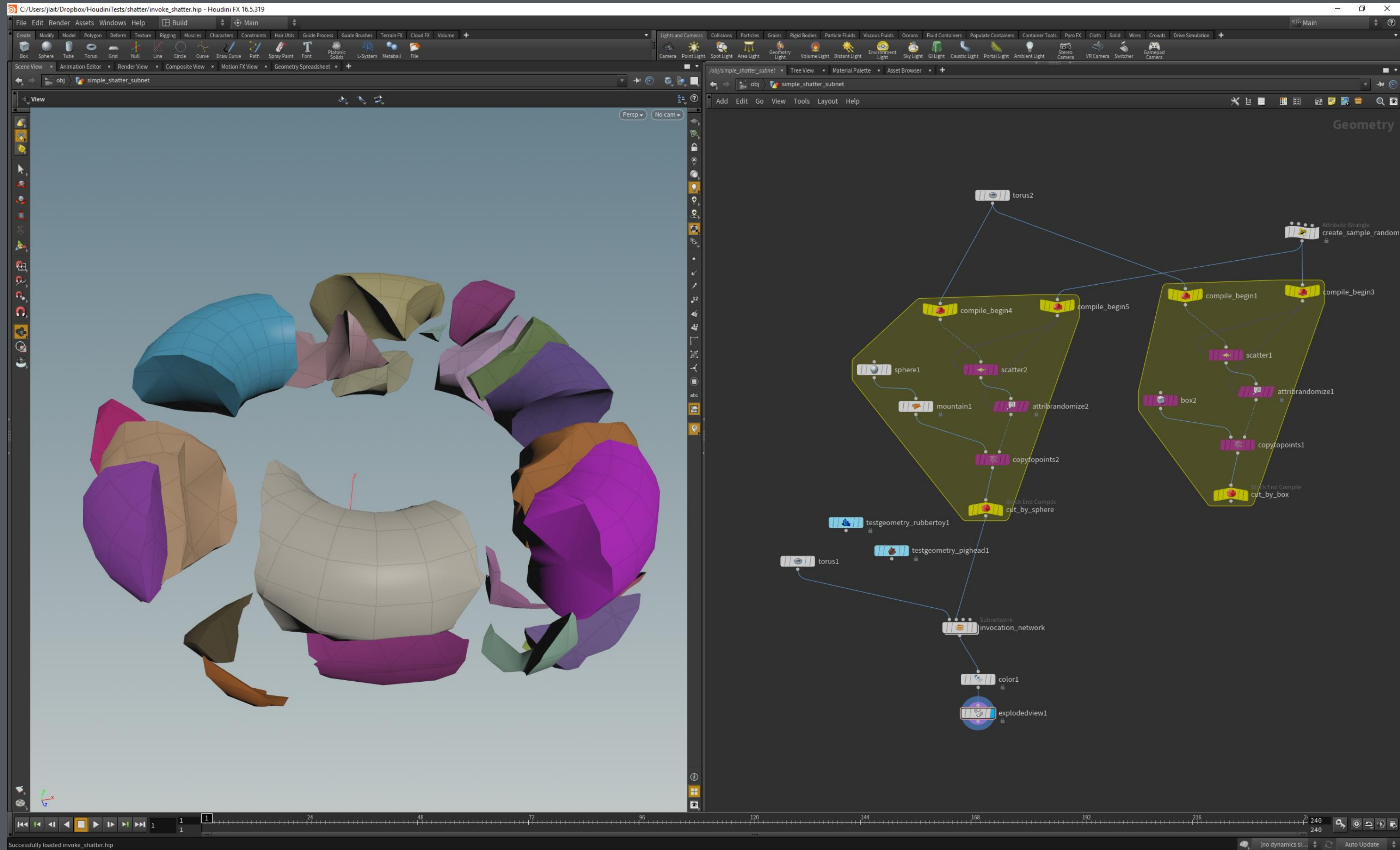
```
jlait@godel ~  
$ dir  
bin  bugfiles  dev_gba  houdini16.5  temp  
  
jlait@godel ~  
$ pfont -s "Houdini" -f 2 | pextrude -z -d .5 | pbevel -w .1 | pcusp -a 25 > log  
o.bpoly|
```











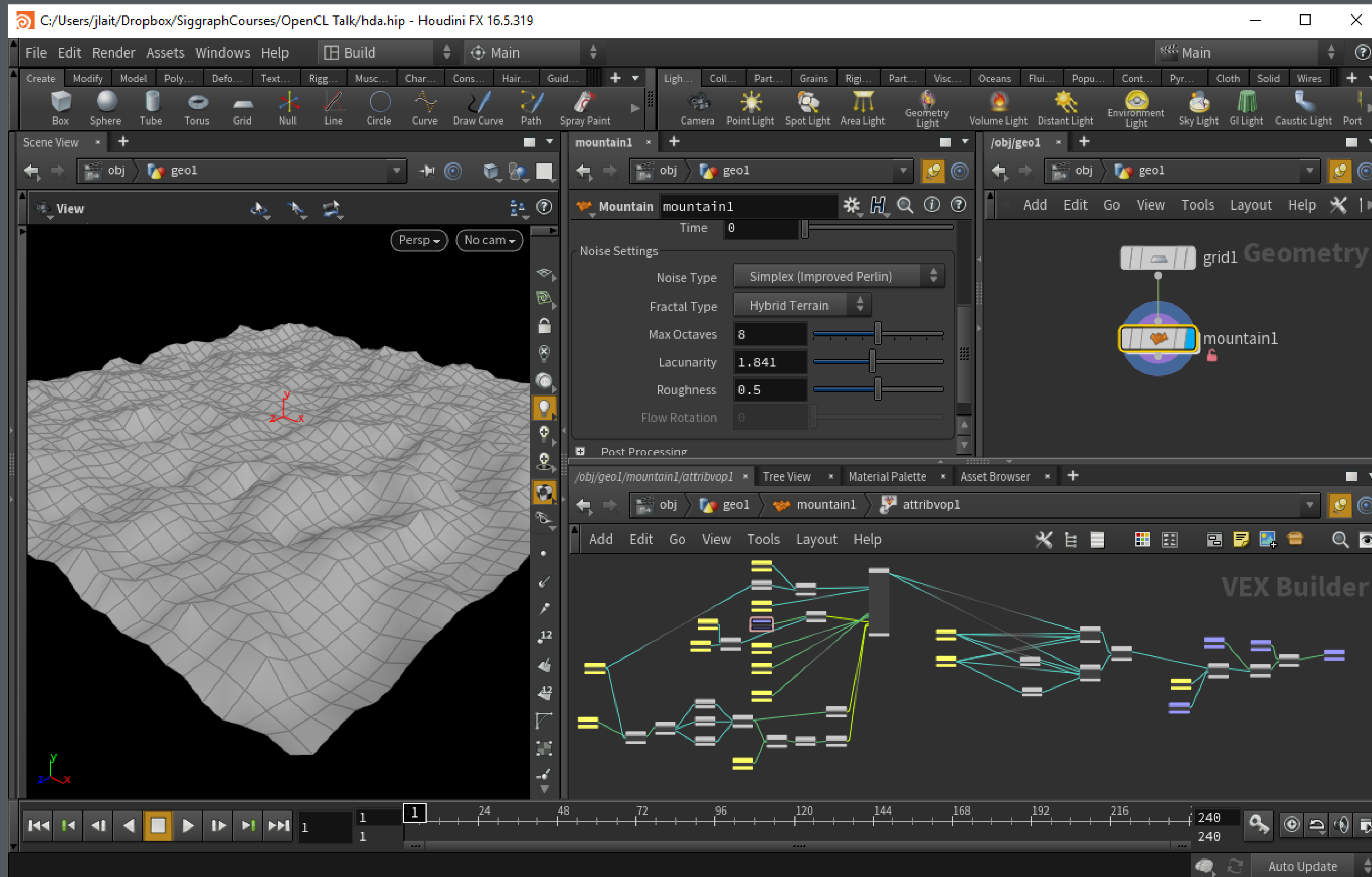
$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} - \nu \nabla^2 \mathbf{u} = \mathbf{g}$$

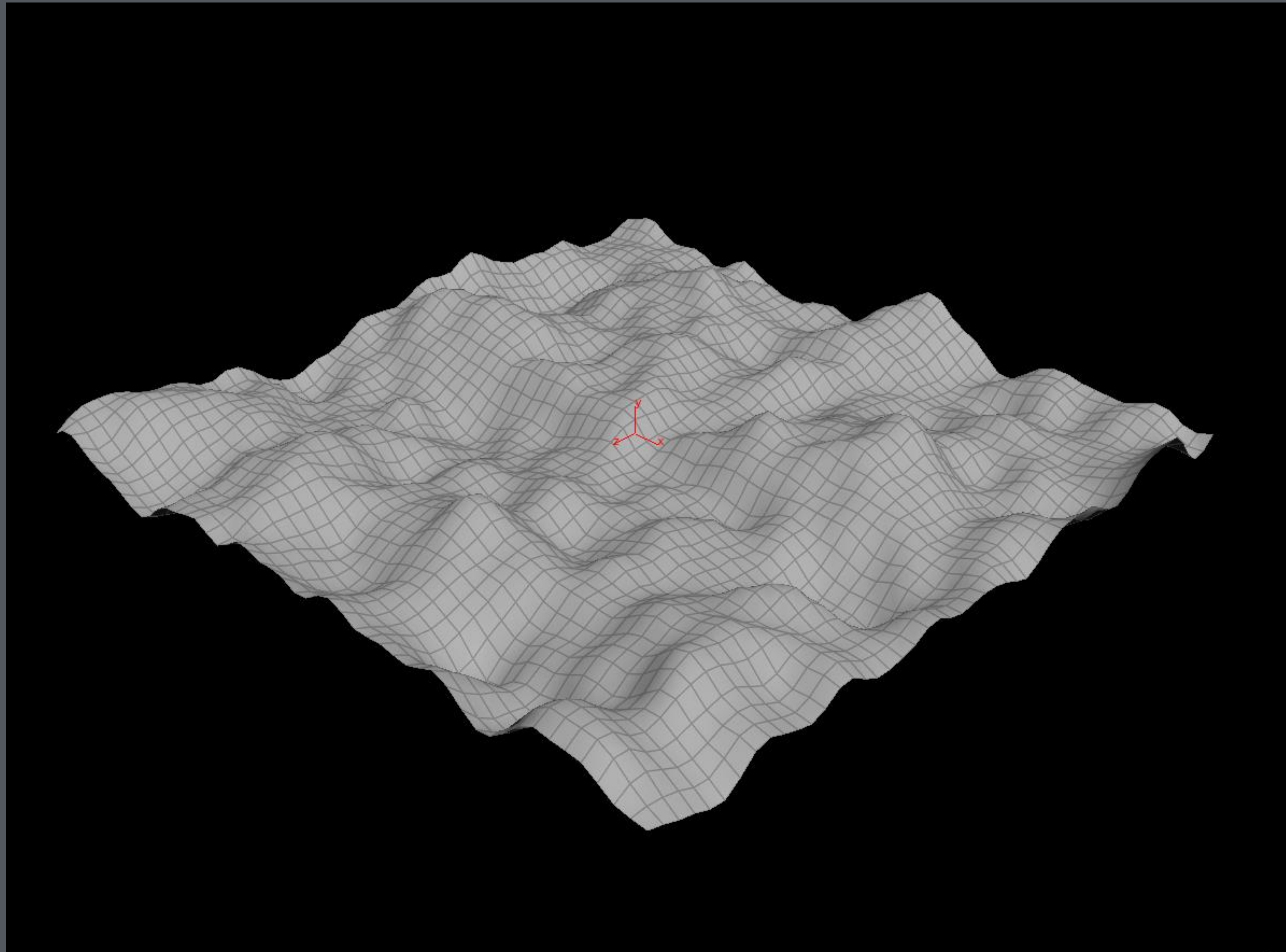
$$\nabla \cdot \mathbf{u} = 0$$

The screenshot shows the Houdini interface with a dynamics network. The network consists of the following nodes:

- emptyobject1 (input)
- scalarfield1 (density)
- vectorfield1 (velocity)
- gasadvect1 (advection term)
- gasbuoyancy1 (buoyancy term)
- gasblur1 (viscosity term)
- gasprojectnondivergent1 (divergence-free constraint)
- multisolver1 (solver)
- output (render output)

 The Navier-Stokes equations are overlaid on the network, with mathematical symbols corresponding to the physical terms being simulated.





```
sop
mountain(int frac_depth=3;           // Fractal depth
         float rough=.6;             // Fractal roughness
         float height=1;             // Height amount

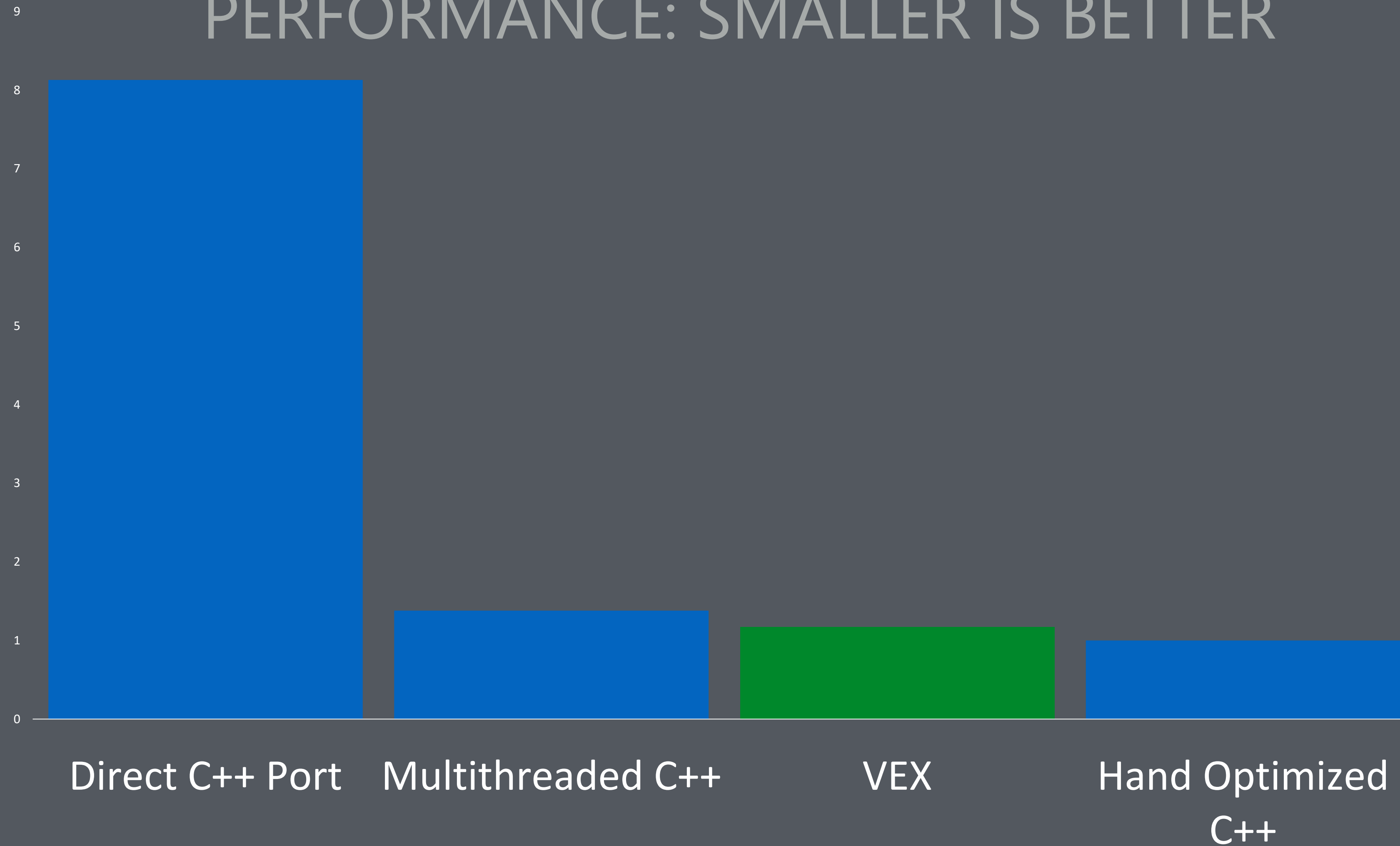
         vector freq=1;              // Noise Frequency
         vector offset=0;            // Noise Offset
         string ntype="perlin";
         int do_color=0;
         float clr_off=0.4;
         float clr_scale=1;
)
{
    int i;
    float scale;
    float nval;
    vector PP = (P + offset) * freq;

    nval = 0;
    scale = height;
    for (i = frac_depth; i-- > 0; scale *= rough)
    {
        if (ntype == "alligator") nval += scale*anoise(PP);
        else if (ntype == "sparse") nval += scale*snoise(PP);
        else nval += scale*noise0(PP);
        PP *= 2;
    }

    // Move in the direction of the surface normal
    P += normalize(N)*nval;

    if (do_color)
    {
        nval = (nval+clr_off)*clr_scale;
        if (nval < 0)
        {
            Cd = cspline(-nval,
                { 0.1, 0.8, 1.0 },
                { 0.1, 0.8, 1.0 },
                { 0.1, 0.7, 1.0 },
                { 0.1, 0.5, 1.0 },
                { 0.1, 0.2, 1.0 },
                { 0.1, 0.1, 1.0 },
                { 0.1, 0.1, 0.7 },
                { 0.1, 0.1, 0.3 },
                { 0.1, 0.1, 0.3 });
        }
        else
        {
            Cd = cspline(nval,
                { 0.4, 0.8, 0.2 },
                { 0.4, 0.8, 0.2 },
                { 0.5, 0.7, 0.2 },
                { 0.5, 0.5, 0.2 },
                { 0.5, 0.2, 0.5 },
                { 0.7, 0.1, 0.6 },
                { 0.8, 0.1, 0.7 },
                { 0.8, 0.8, 0.8 },
                { 0.8, 0.8, 0.8 });
        }
    }
}
}
```

PERFORMANCE: SMALLER IS BETTER



The screenshot displays the Houdini interface with a 3D scene view on the left and an Attribute Wrangle node editor on the right. The scene view shows a grid of points with a wavy heightmap, colored in shades of green and blue. The Attribute Wrangle node is named 'attribwrangle1' and contains the following VEX code:

```
@P.y += noise(@P / ch('element_size')) * ch('amplitude');  
@Cd = chrap('height_ramp', @P.y);
```

The node parameters are set as follows:

- Group: (empty)
- Group Type: Guess from Group
- Run Over: Points
- Element Size: 1
- Amplitude: 0.946
- Height Ramp: A color ramp from blue to white, with a point number of 5, position 1, and color (1, 1, 1). Interpolation is set to Linear.

The bottom right corner of the scene view shows performance metrics: 20fps, 49.46ms, 9,801 prims, and 10,000 points.

The screenshot displays the Houdini software interface. The central viewport shows a 3D model of a creature with a grid overlay. On the right, the Attribute Wrangle node is open, showing the following VEX code:

```
int npts[];  
npts = neighbours(0, @ptnum);  
  
vector avg = 0;  
foreach (int npt; npts)  
{  
    vector npos = point(0, 'P', npt);  
    avg += npos;  
}
```

The interface also shows a timeline at the bottom with a play button and a performance monitor displaying 99fps, 10.15ms, 20,421 prims, and 20,408 points.

OpenCL vs VEX

The image shows a screenshot of the Houdini software interface. On the left, the 'Dynamics' panel is open, displaying the 'Gas Disturbance CL' node. The node's parameters are as follows:

- Disturb Field: [Empty]
- Threshold Field: [Empty]
- Disturbance: 0
- Time Scale: 1
- Cutoff: 0.2
- Locality: 1
- Use Block Size: [Checked]
- Block Size: 0.1
- Use OpenCL: [Checked]
- Use Control Field: [Checked]
- Control Field: [Empty]
- Control Influence: 0.5
- Control Min: 0
- Control Max: 1
- Default Operation: Set Initial
- Make Objects Mutual Affectors: [Checked]
- Group: *
- Data Name: \$OS
- Unique Data Name: [Checked]
- Solver Per Object: [Unchecked]

On the right, a VIM editor window displays the OpenCL code for the 'GAS_DisturbFieldCL.C' file. The code is as follows:

```

control_par.y = controlMin;
control_par.z = 1.0f / (controlMax - controlMin);

// Sizes for blocks
orig.x = orig_(0); orig.y = orig_(1); orig.z = orig_(2);
size.x = size_(0); size.y = size_(1); size.z = size_(2);

CE_Context *context = CE_Context::getContext();
cl::Program prog = context->loadProgram("sim/disturbField.cl",
progflags.buffer());

cl::kernel kern = context->loadkernel(prog, "disturbField");

// Set kernel Arguments
int argidx = 0;
kern.setArg(argidx++, dist.buffer() );
kern.setArg(argidx++, threshold.buffer() );
kern.setArg(argidx++, cutoff );
kern.setArg(argidx++, coeff );
kern.setArg(argidx++, frame );

if( control )
{
    kern.setArg(argidx++, control->buffer());
    kern.setArg(argidx++, control_par );
}

if( locality_inv < 0.0f )
{
    kern.setArg(argidx++, blocksize_inv );
    kern.setArg(argidx++, orig );
    kern.setArg(argidx++, size );
} else
{
    kern.setArg(argidx++, locality_inv );
}

kern.setArg(argidx++, dist.getOffset());
kern.setArg(argidx++, stride );

cl::kernelFunctor func = kern.bind( CE_Context::getContext()->getQueue(),
func());
    
```

The VIM editor window also shows the file path: `GAS_DisturbFieldCL.C (~/.dev_green/src/houdini/lib/H_SIM/GAS) - VIM`. The status bar at the bottom right of the VIM window shows `378,1` and `51%`.

The screenshot displays the Houdini interface with the Dynamics node editor on the left, the Gas OpenCL shreddingCL node settings in the center, and a VIM editor window on the right showing the gasShred.cl kernel code.

Dynamics Node Editor: Shows a network of nodes including "Gas Field VOP", "calculate_shredding", "Gas OpenCL shreddingCL", and "Switch Shred_CL_Switch".

Gas OpenCL shreddingCL Node Settings:

- Kernel Name: **gasShred**
- Kernel File: **sim/gasShred.cl**
- Kernel Options: **-DCLIP**
- Buttons: Generate Kernel, Recompile Kernel

VIM Editor (gasShred.cl):

```

kernel void gasshred(
    int stride_x,
    int stride_y,
    int stride_z,
    int stride_offset,
    float  PushPullThreshold ,
    float  PushAmount ,
    float  PullAmount ,
    float  Fade ,
    float  Clipvalue ,
    float  ControlInfluence ,
    float  ControlMax ,
    float  ControlMin ,
    global float * TemperatureField ,
    global float * GradientField_1 ,
    global float * GradientField_2 ,
    global float * GradientField_3 ,
    global float * ControlField )
{
    int gidx = get_global_id(0);
    int gidy = get_global_id(1);
    int gidz = get_global_id(2);

    int idx = stride_offset + stride_x * gidx
                + stride_y * gidy
                + stride_z * gidz;

    float grad1      = GradientField_1[idx];
    float grad2      = GradientField_2[idx];
    float grad3      = GradientField_3[idx];
    float temperature = TemperatureField[idx];
    float scaler;
    float sum;

#ifdef CLIP
    // normalize gradient
    float grad_norm = sqrt( grad1 * grad1 + grad2 * grad2 + grad3 * grad3 );
    grad_norm = max( 1e-6f, grad_norm );

    grad1 = grad1 / grad_norm;
    grad2 = grad2 / grad_norm;
    grad3 = grad3 / grad_norm;

    // clamp gradient length
    grad_norm = clamp(grad_norm, 0.0f, clipvalue);

    // Restore gradient
    "gasshred.cl" 92L, 2695c
    
```

The screenshot displays the Houdini interface with a sine wave effect applied to a grid object. The interface includes a menu bar, toolbars, and several viewports. The main viewport shows a perspective view of a grid with a sine wave deformation. The right-hand side features a 'Geometry' panel with a tree view showing the 'grid1' object and an 'openc1' node. The 'openc1' node is expanded to show the kernel code for the 'sineeffect' kernel.

```
kernel void sineeffect(  
    int P_length,  
    global float * P,  
    float amplitude ,  
    float period ,  
    float phase  
)  
{  
    int idx = get_global_id(0);  
    if (idx >= P_length)  
        return;  
  
    float3 pos = vload3(idx, P);  
  
    pos.y += amplitude *  
        sin( length(pos) / period + phase );  
  
    vstore3(pos, idx, P);  
}
```

Kernel Options:

- Use Write Back Kernel
-
- Recompile Kernel

The screenshot displays the Houdini interface for editing an OpenCL kernel. The main window is titled "C:/Users/jlait/Dropbox/SiggraphCourses/OpenCL Talk/sineeffect.hip - Houdini FX 16.5.319".

Kernel Code:

```
kernel void sineeffect(  
    int P_length,  
    global float * P,  
    float amplitude ,  
    float period ,  
    float phase  
)  
{  
    int idx = get_global_id(0);  
    if (idx >= P_length)  
        return;  
  
    float3 pos = vload3(idx, P);  
  
    pos.y += amplitude *  
        sin( length(pos) / period + phase );  
  
    vstore3(pos, idx, P);  
}
```

Parameter Bindings:

- Parameter Name:** P
Options:
 - Parameter Type: Attribute
 - Attribute: P
 - Class: Point
 - Type: Float
 - Size: 3
 - Readable:
 - Writeable:
 - Optional:
- Parameter Name:** amplitude
Options:
 - Parameter Type: Float
 - Float: 1
 - Time Scale: None
- Parameter Name:** period
Options:

The interface also shows a timeline at the bottom with a play button and a search bar. The SideFX logo is visible in the bottom right corner.

The screenshot displays the Houdini software interface with two panels open for editing an OpenCL kernel named 'sineeffect'.

Left Panel (Kernel Editor):

- Kernel Name:** sineeffect
- Use Code Snippet:**
- Kernel Code:**

```
kernel void sineeffect(
    int P_length,
    global float * P,
    float amplitude,
    float period,
    float phase
)
{
    int idx = get_global_id(0);
    if (idx >= P_length)
        return;

    float3 pos = vload3(idx, P);

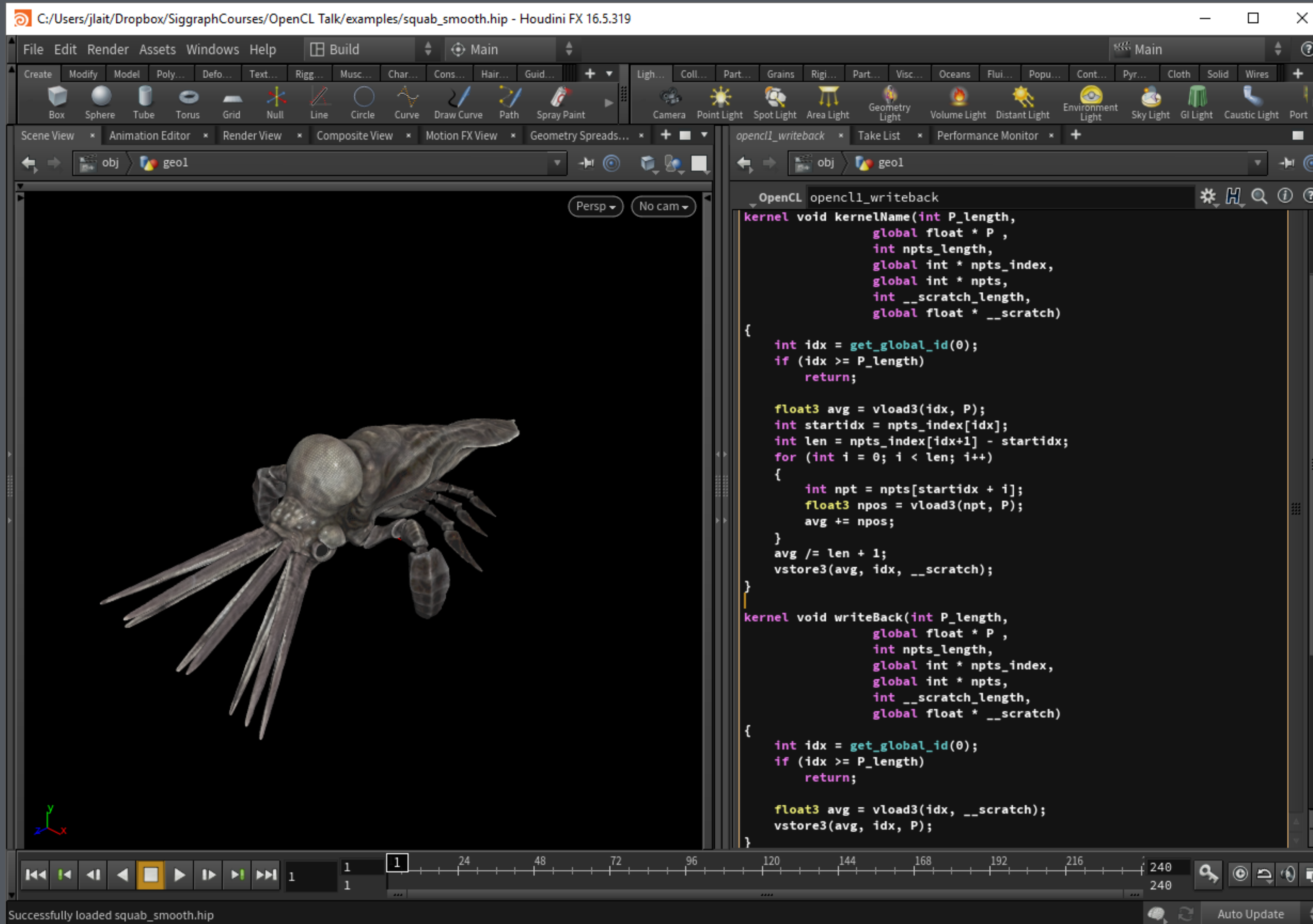
    pos.y += amplitude *
        sin( length(pos) / period + phase );

    vstore3(pos, idx, P);
}
```
- Kernel Options:**
 - Use Write Back Kernel
 -
 - Recompile Kernel

Right Panel (Performance Monitor):

- Run Over:** First Writeable Attribute
- Iterations:** 1 (with a slider)
- Finish Kernels
- Include Time
- Include Timestep
- Timescale:** 1 (with a slider)
- Include Simplex Noise Data

The bottom of the interface shows a timeline with a play button highlighted, indicating the kernel is being executed or about to be executed. The timeline has markers at 1, 24, 39, 48, 72, 96, 120, 144, 168, 192, 216, 240.



The screenshot displays the Houdini software interface. On the left, a 3D scene view shows a rendered squid-like creature. On the right, the OpenCL kernel editor is open, showing two kernel functions: `kernelName` and `writeBack`. The `kernelName` function calculates the average position of points within a specified range. The `writeBack` function writes the calculated average back to the point's position in the point cloud.

```
kernel void kernelName(int P_length,
    global float * P ,
    int npts_length,
    global int * npts_index,
    global int * npts,
    int __scratch_length,
    global float * __scratch)
{
    int idx = get_global_id(0);
    if (idx >= P_length)
        return;

    float3 avg = vload3(idx, P);
    int startidx = npts_index[idx];
    int len = npts_index[idx+1] - startidx;
    for (int i = 0; i < len; i++)
    {
        int npt = npts[startidx + i];
        float3 npos = vload3(npt, P);
        avg += npos;
    }
    avg /= len + 1;
    vstore3(avg, idx, __scratch);
}

kernel void writeBack(int P_length,
    global float * P ,
    int npts_length,
    global int * npts_index,
    global int * npts,
    int __scratch_length,
    global float * __scratch)
{
    int idx = get_global_id(0);
    if (idx >= P_length)
        return;

    float3 avg = vload3(idx, __scratch);
    vstore3(avg, idx, P);
}
```

The screenshot displays the Houdini interface for a file named `C:/Users/jlait/Dropbox/SiggraphCourses/OpenCL Talk/examples/squab_smooth.hip`. The main viewport shows a 3D render of a squid-like creature in a perspective view. The right-hand pane, titled "Geometry", shows a node network for the object `obj/geo1`. The network consists of the following nodes:

- `compile_begin3` (yellow node)
- `Block Begin repeat_begin5` (orange node)
- `Feedback` (blue node)
- `opengl1_writeback2` (white node)
- `Block End repeat_end5` (orange node)
- `Feedback` (blue node)
- `compile_end3` (yellow node)

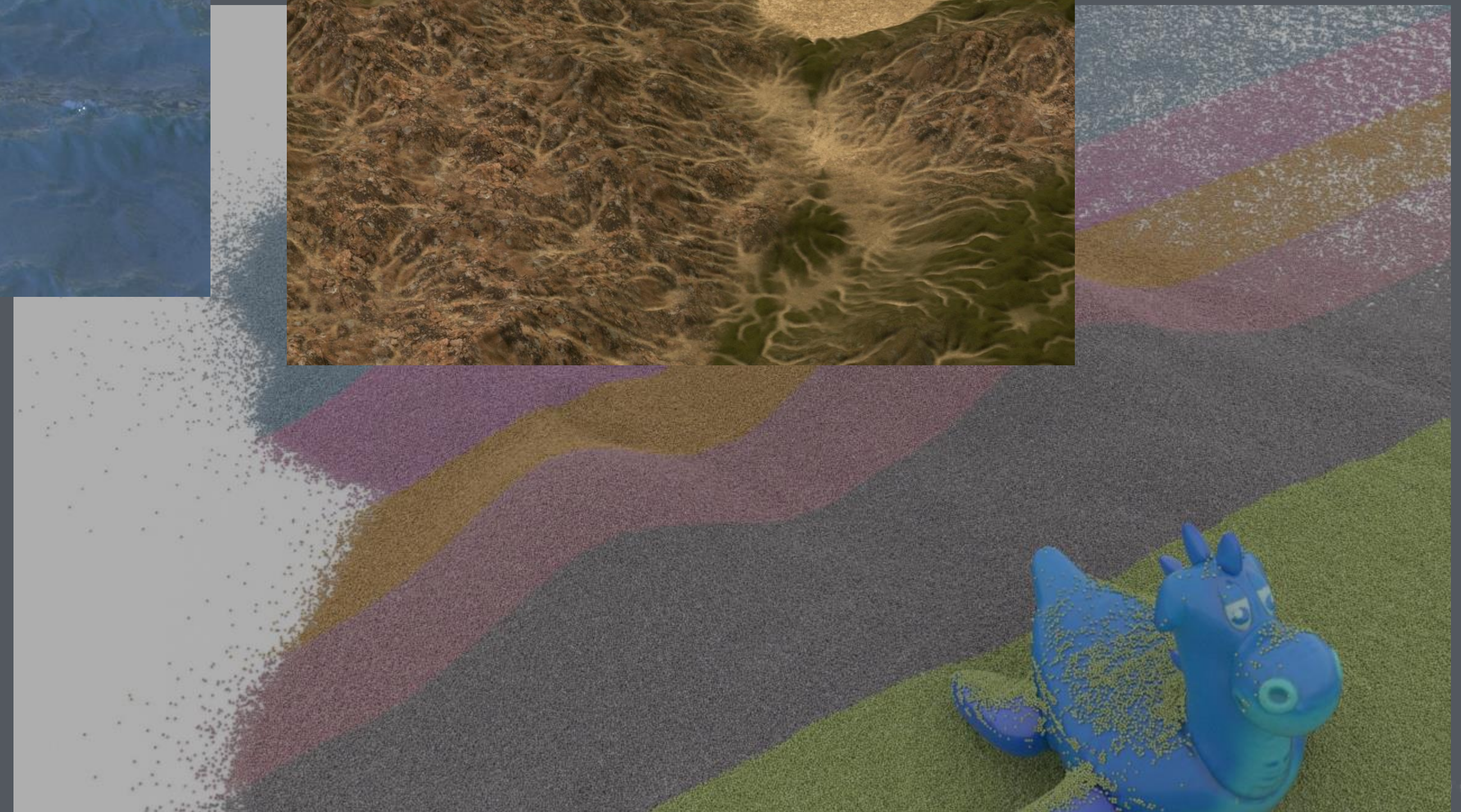
The network is enclosed in a yellow dashed box. The bottom of the interface features a timeline with a play button and a status bar that reads "Successfully loaded squab_smooth.hip". The SideFX logo is visible in the bottom right corner.

```
/// @{
/// The attribute data may be moved to the GPU, and hence backed
/// by a compute buffer (GA_CEAtribute). Flushing will delete
/// our buffer & copy it back from the GPU if it has been marked
/// as modified.
///
/// Caches a CE Attribute. If a cache already exists, the cached
/// data is returned and the write flag updated. If the cache
/// does not exist, an attempt is made to build a buffer. If
/// read is true, the buffer is initialized with the geometry data.
/// This can return NULL if a failure to build an attribute occurs
/// (for example, unsupported type or no GPU)
GA_CEAtribute *getCEAttribute(GA_StorageClass storage, int &tuplesize,
bool isarray, bool read, bool write);
///
/// Any CE cache which was marked as written to will be copied
/// back to the CPU. It will be left on the GPU, however.
void flushCEwriteCaches(bool clearwriteback=true);
bool hasPendingCEwriteBack() const;
///
/// Remove all CE caches, copying back any marked as written to.
void flushCECaches();
/// @}
```

The screenshot displays the Houdini software interface with the following components:

- Scene View:** Shows a 3D perspective view of a torus object with a grid overlay, indicating a workset-based deformation or simulation.
- Parameters Panel:** Displays the 'gauss_seidel' kernel parameters:
 - Run Over: Detail Attribute of Wor...
 - Iterations: 10
 - Worksets Begin Attr.: workset_begin
 - Worksets Length Attr.: workset_len
 - Finish Kernels:
 - Include Time:
 - Include Timestep:
 - Include Simplex Noise Data:
 - Timescale: 1
- Kernel Code Editor:** Shows the OpenCL kernel code for 'gauss_seidel':

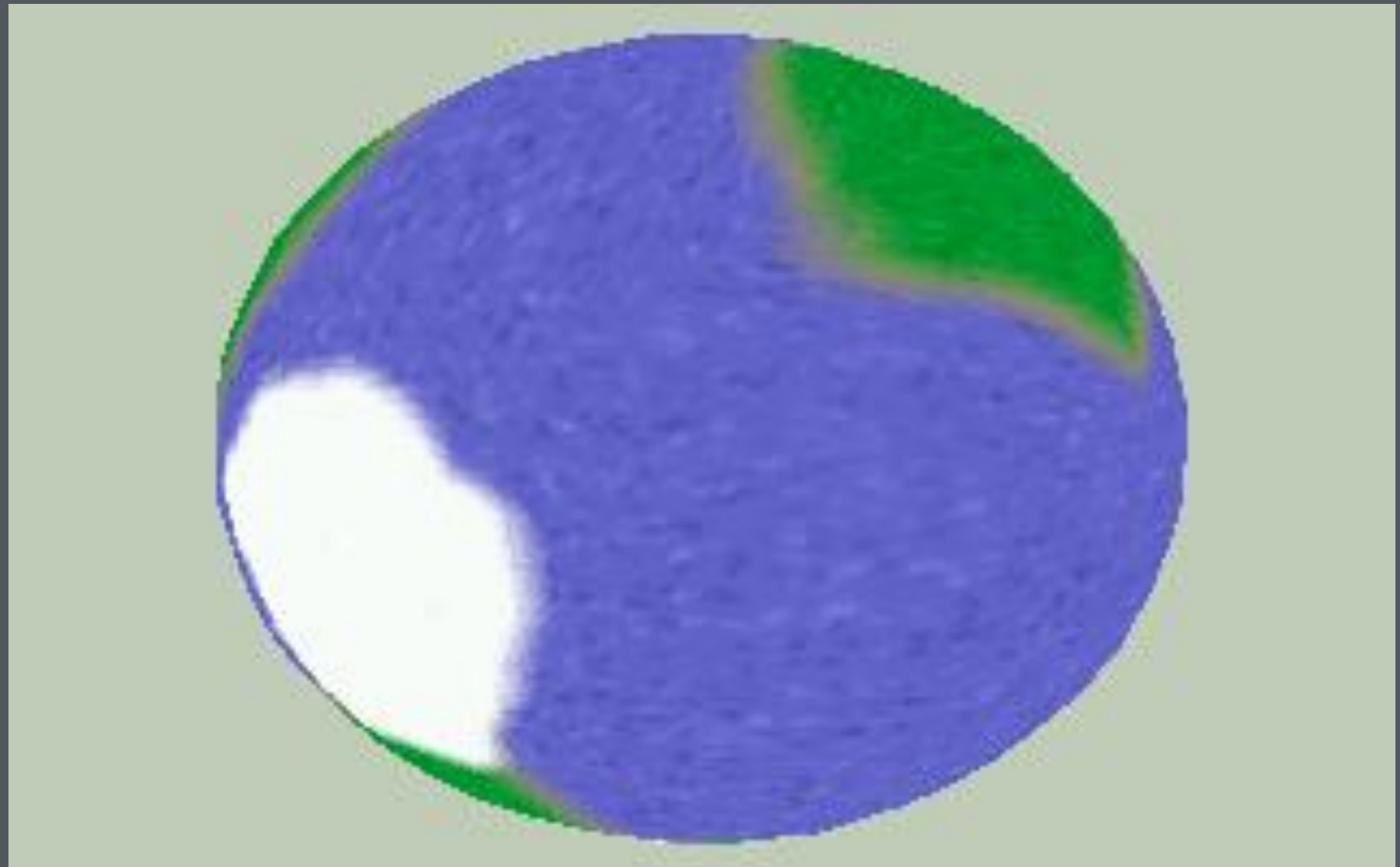
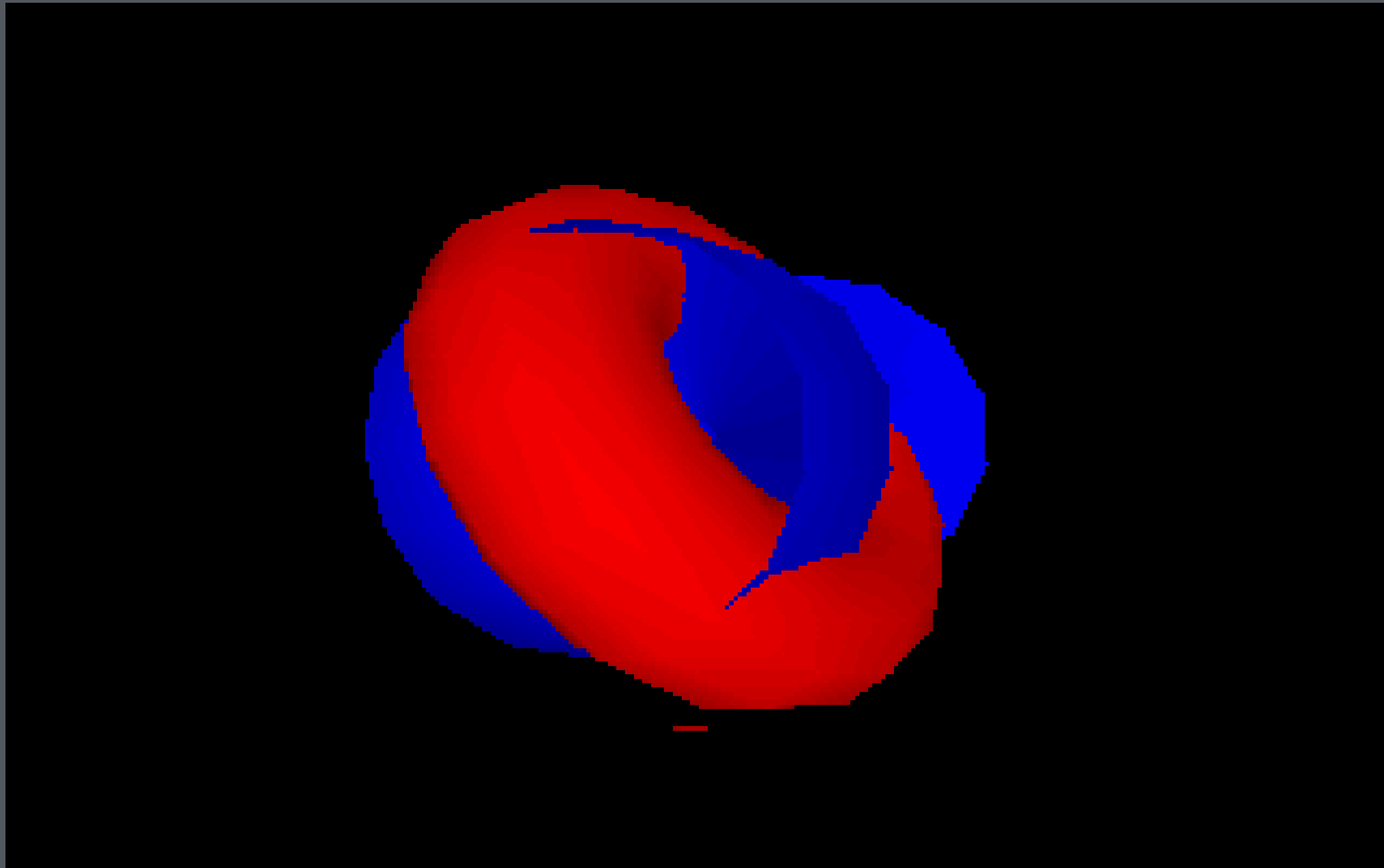
```
kernel void kernelName(  
    int worksets_begin,  
    int worksets_length,  
    int P_length,  
    global float * P ,  
    int restlength_length,  
    global float * restlength ,  
    int points_length,  
    global int * points_index,  
    global int * points  
)  
{  
    int idx = get_global_id(0);  
    if (idx >= worksets_length)  
        return;  
    idx += worksets_begin;  
  
    int p0idx = points[points_index[idx]];  
    int p1idx = points[points_index[idx]+1];  
  
    float3 p0 = vload3(p0idx, P);  
    float3 p1 = vload3(p1idx, P);  
  
    float actuallen = length(p0-p1);  
    actuallen = max(actuallen, 0.001f);  
    float rest = restlength[idx];  
  
    float change = rest - actuallen;  
    float3 displace = (p0 - p1) * change  
        / actuallen;  
  
    displace /= 2;  
    p0 += displace;  
    p1 -= displace;  
  
    vstore3(p0, p0idx, P);  
    vstore3(p1, p1idx, P);  
}
```
- Timeline:** Shows a timeline with a current frame of 1 and a total of 240 frames.
- Status Bar:** Displays 'Successfully saved workset.hip (Sun May 6 11:37:23 2018)' and 'Auto Update'.



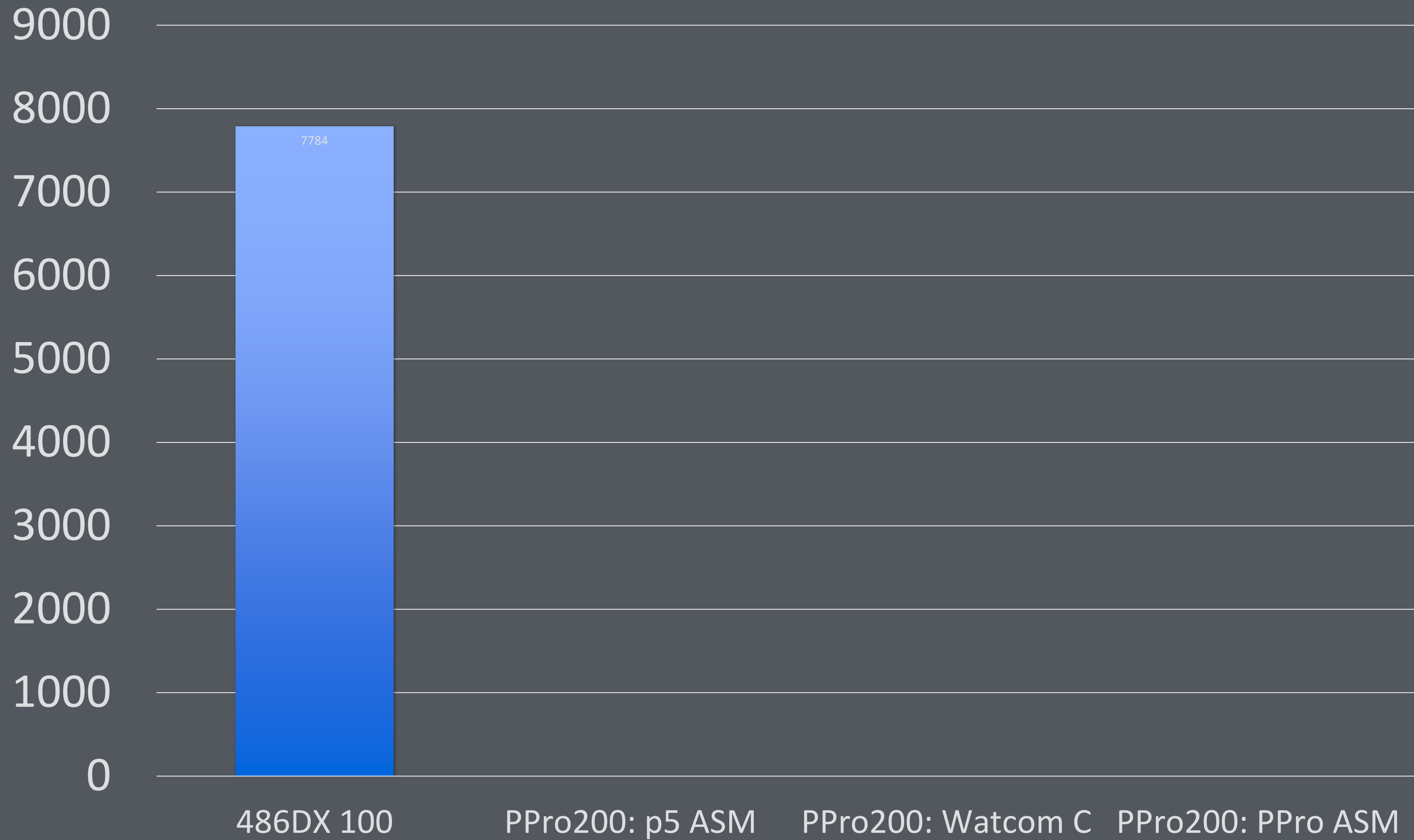
```
kernel void smooth(int P_length,
                  global float * P ,
                  int npts_length,
                  global int * npts_index,
                  global int * npts,
                  int __scratch_length,
                  global float * __scratch)
{
    int idx = get_global_id(0);
    if (idx >= P_length)
        return;

    float3 avg = 0;
    int startidx = npts_index[idx];
    int len = npts_index[idx+1] - startidx;
    for (int i = 0; i < len; i++)
    {
        int npt = npts[startidx + i];
        float3 npos = vload3(npt, P);
        avg += npos;
    }
    avg /= len;
    vstore3(avg, idx, __scratch);
}
```

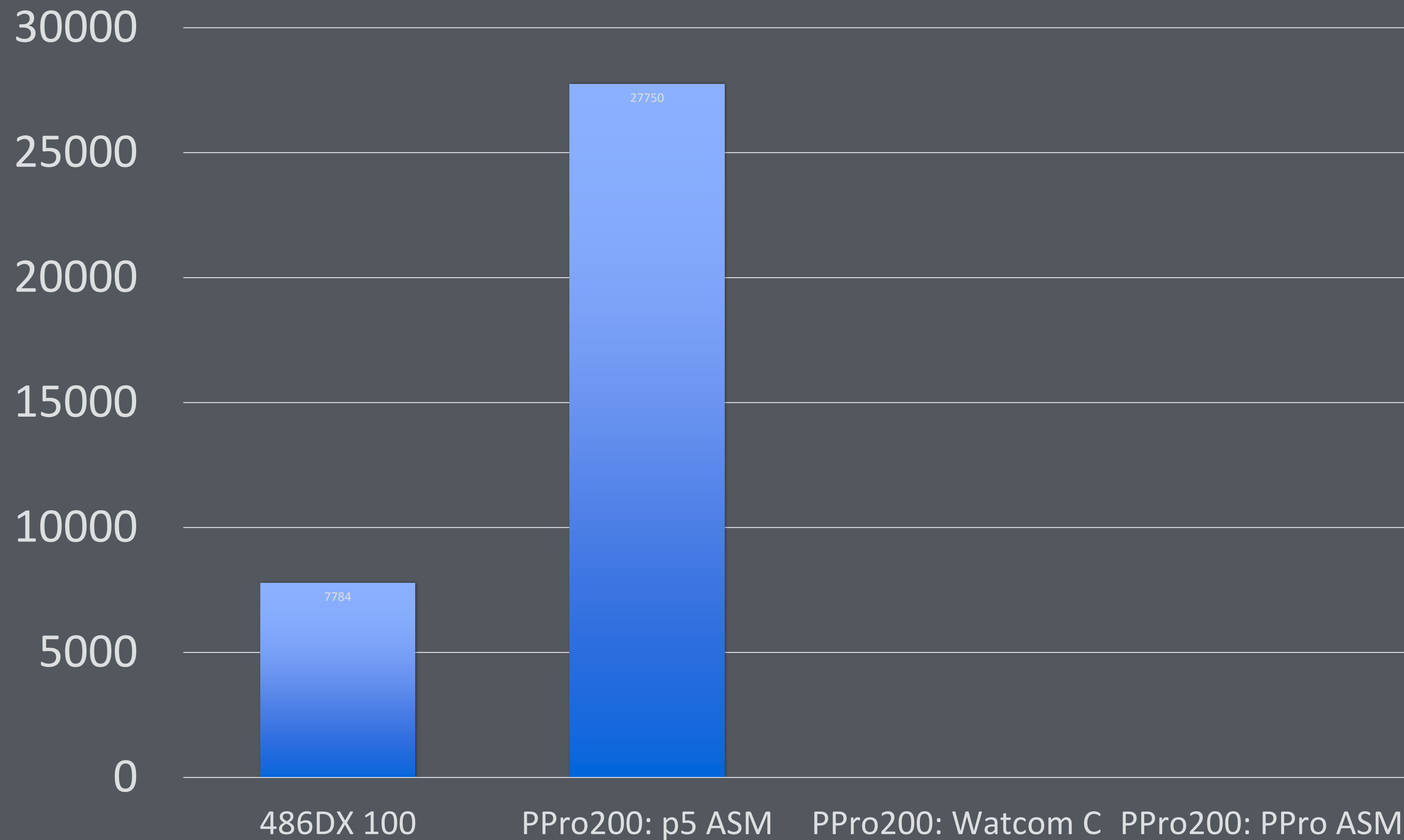
1. Performance Portable



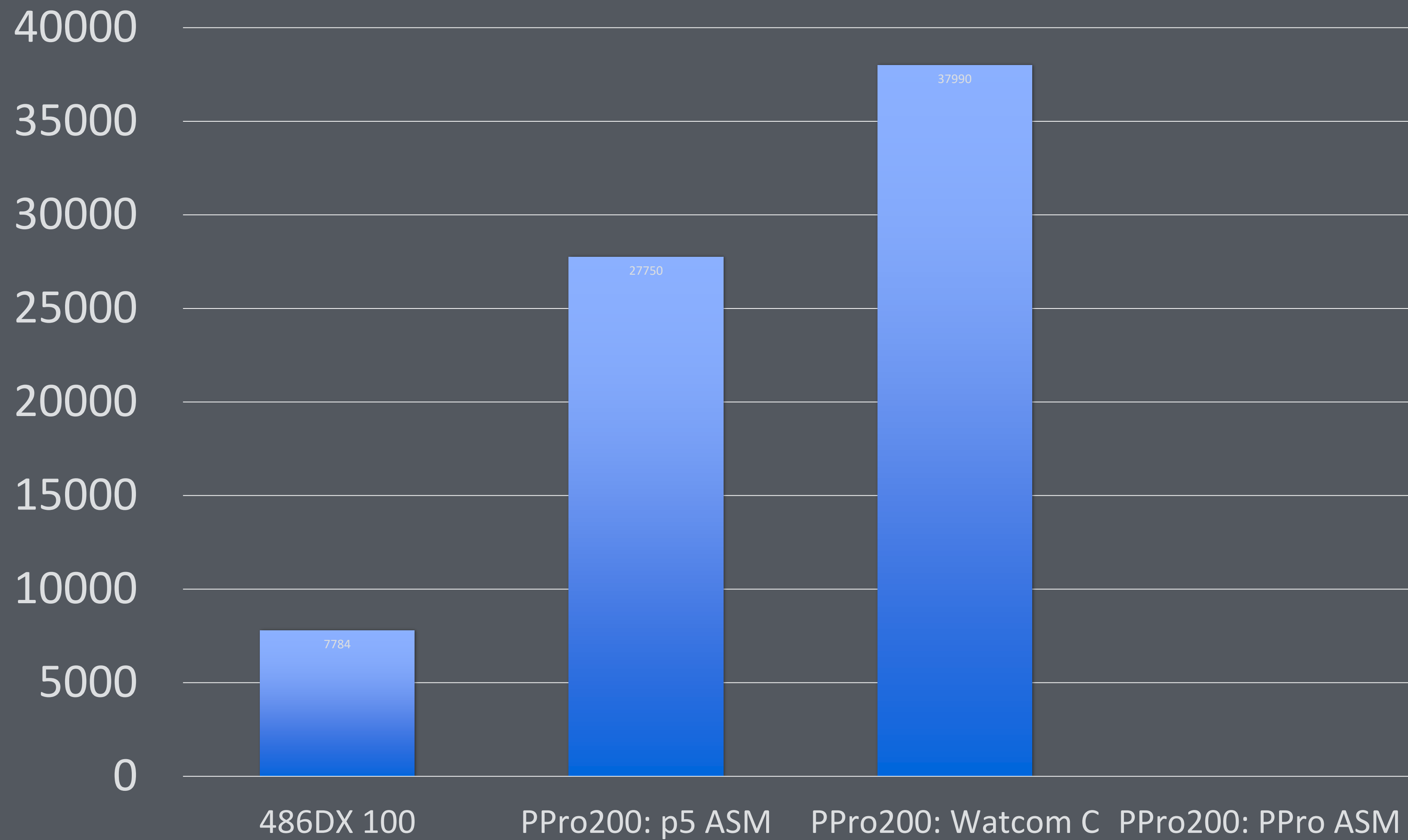
Poly Per Second



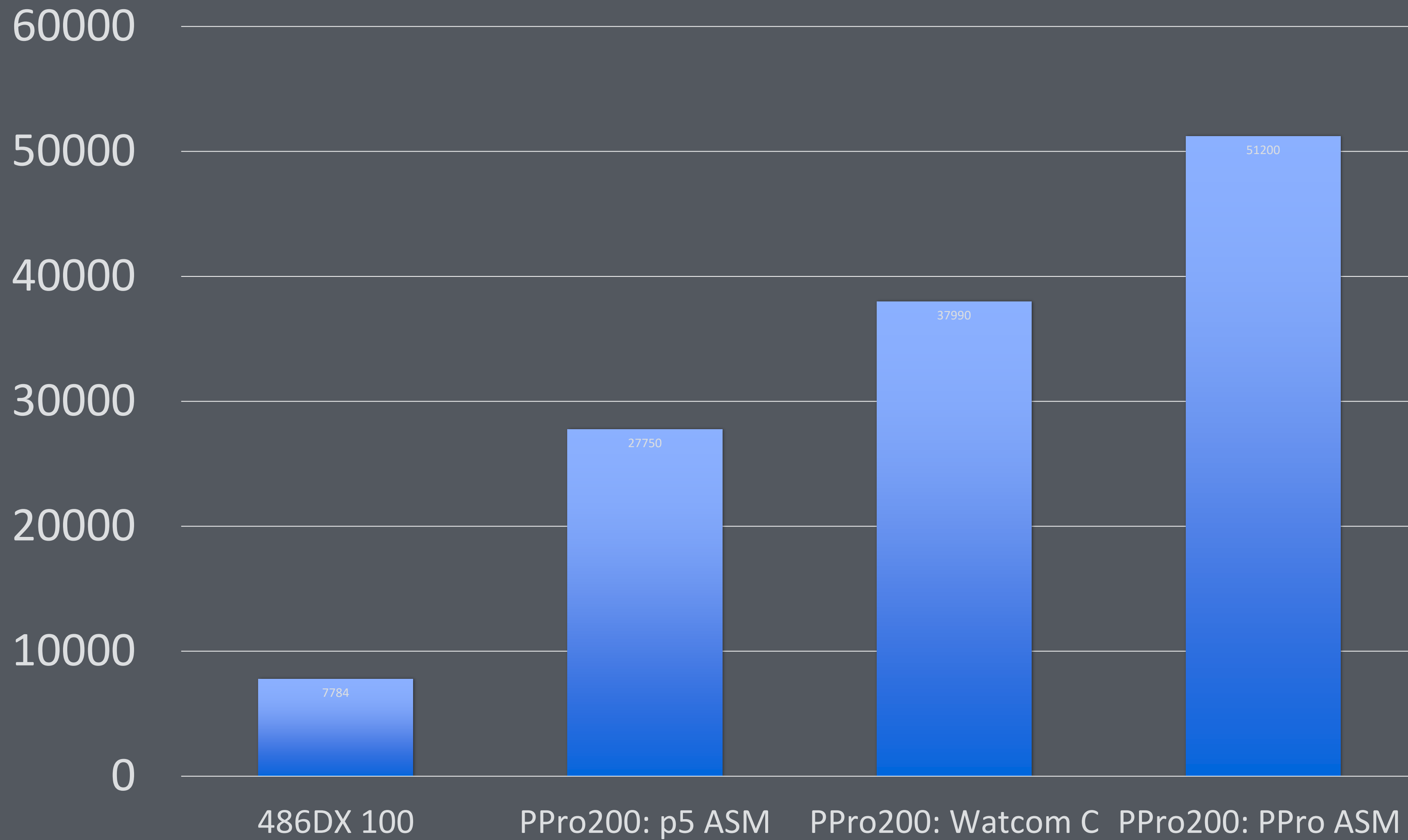
Poly Per Second



Poly Per Second



Poly Per Second



```
kernel void smooth(int P_length,
                  global float * P ,
                  int npts_length,
                  global int * npts_index,
                  global int * npts,
                  int __scratch_length,
                  global float * __scratch)
{
    int idx = get_global_id(0);
    if (idx >= P_length)
        return;

    float3 avg = 0;
    int startidx = npts_index[idx];
    int len = npts_index[idx+1] - startidx;
    for (int i = 0; i < len; i++)
    {
        int npt = npts[startidx + i];
        float3 npos = vload3(npt, P);
        avg += npos;
    }
    avg /= len;
    vstore3(avg, idx, __scratch);
}
```

1. Performance Portable


```
kernel void smooth(int P_length,
                  global float * P ,
                  int npts_length,
                  global int * npts_index,
                  global int * npts,
                  int __scratch_length,
                  global float * __scratch)
{
    int idx = get_global_id(0);
    if (idx >= P_length)
        return;

    float3 avg = 0;
    int startidx = npts_index[idx];
    int len = npts_index[idx+1] - startidx;
    for (int i = 0; i < len; i++)
    {
        int npt = npts[startidx + i];
        float3 npos = vload3(npt, P);
        avg += npos;
    }
    avg /= len;
    vstore3(avg, idx, __scratch);
}
```

1. Performance Portable
2. Default Initializers

```
kernel void smooth(int P_length,
                  global float * P ,
                  int npts_length,
                  global int * npts_index,
                  global int * npts,
                  int __scratch_length,
                  global float * __scratch)
{
    int idx = get_global_id(0);
    if (idx >= P_length)
        return;

    float3 avg = 0;
    int startidx = npts_index[idx];
    int len = npts_index[idx+1] - startidx;
    for (int i = 0; i < len; i++)
    {
        int npt = npts[startidx + i];
        float3 npos = vload3(npt, P);
        avg += npos;
    }
    avg /= len;
    vstore3(avg, idx, __scratch);
}
```

1. Performance Portable
2. Default Initializers
3. Safe Operations

```
kernel void tumbleMaterial(  
    float flow_rate,  
    float repose_angle,  
    float height_factor,  
    int stride_x,  
    int stride_y,  
    int stride_z,  
    int stride_offset,  
    float voxelsize_x,  
    float voxelsize_y,  
    float voxelsize_z,  
    global float * height,  
    global float * material,  
#ifdef CALC_FLOW  
    global float * flow_x,  
    global float * flow_y,  
#endif  
    global float * write_back  
#ifdef HAS_entrained  
    , global float * entrained  
#endif  
    , global float * write_back_entrained  
    , float entrainmentrate  
    , int openborder  
)  
{  
    int gidx = get_global_id(0);  
    int gidy = get_global_id(1);  
    int gidz = get_global_id(2);  
  
    int idx = stride_offset + stride_x * gidx  
                + stride_y * gidy  
                + stride_z * gidz;  
  
    int bound_x = get_global_size(0);  
    int bound_y = get_global_size(1);  
}
```

1. Performance Portable
2. Default Initializers
3. Safe Operations
4. Optional Bindings

```
kernel void tumbleMaterial(  
    float flow_rate,  
    float repose_angle,  
    float height_factor,  
    int stride_x,  
    int stride_y,  
    int stride_z,  
    int stride_offset,  
    float voxelsize_x,  
    float voxelsize_y,  
    float voxelsize_z,  
    global float * height,  
    global float * material,  
#ifdef CALC_FLOW  
    global float * flow_x,  
    global float * flow_y,  
#endif  
    global float * write_back  
#ifdef HAS_entrained  
    , global float * entrained  
#endif  
    , global float * write back entrained  
    , float entrainmentrate  
    , int openborder  
)  
{  
    int gidx = get_global_id(0);  
    int gidy = get_global_id(1);  
    int gidz = get_global_id(2);  
  
    int idx = stride_offset + stride_x * gidx  
                + stride_y * gidy  
                + stride_z * gidz;  
  
    int bound_x = get_global_size(0);  
    int bound_y = get_global_size(1);
```

1. Performance Portable
2. Default Initializers
3. Safe Operations
4. Optional Bindings
5. Default Values

```
kernel void tumblematerial(  
    float flow_rate,  
    float repose_angle,  
    float height_factor,  
    int stride_x,  
    int stride_y,  
    int stride_z,  
    int stride_offset,  
    float voxelsize_x,  
    float voxelsize_y,  
    float voxelsize_z,  
    global float * height,  
    global float * material,  
#ifdef CALC_FLOW  
    global float * flow_x,  
    global float * flow_y,  
#endif  
    global float * write_back  
#ifdef HAS_entrained  
    , global float * entrained  
#endif  
    , global float * write_back_entrained  
    , float entrainmentrate  
    , int openborder  
)  
{  
    int gidx = get_global_id(0);  
    int gidy = get_global_id(1);  
    int gidz = get_global_id(2);  
  
    int idx = stride_offset + stride_x * gidx  
                + stride_y * gidy  
                + stride_z * gidz;  
  
    int bound_x = get_global_size(0);  
    int bound_y = get_global_size(1);  
}
```

1. Performance Portable
2. Default Initializers
3. Safe Operations
4. Optional Bindings
5. Default Values
6. Parameter List Inspection

THANK YOU

