Implementing an experimental RenderMan compliant REYES renderer

Davide Pasca - dpasca@gmail.com

https://github.com/dpasca/RibTools

2010/4 (rev. 2022/11)

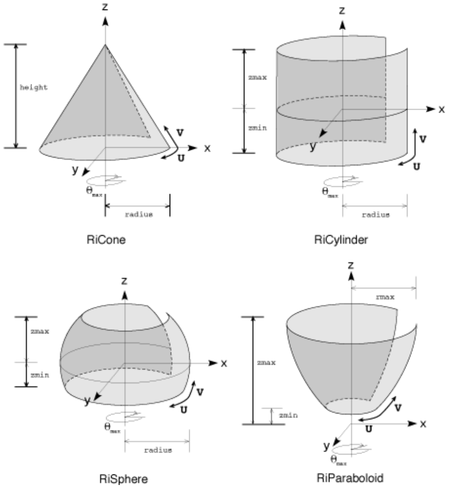
About REYES

- Reyes or REYES (*Renders Everything You Ever Saw*)
- A flexible renderer, developed by Lucasfilm CG div. ("Pixar" from 1986)
- First used in 1984 in *Start Trek II*
- ..still used today in Pixar's *Photorealistic RenderMan*

* Images, from top, are copyright of Paramount Pictures and Pixar (subsidiary of The Walt Disney Company)

RenderMan compliant ?

- Defines a renderer with some basic capabilities such as:
  - A RenderMan graphics state machine
  - Hidden surface elimination
  - Pixel filtering and anti-aliasing
  - User programmable shaders
  - Texture mapping
  - Etc…



RiCone

RiCylinder

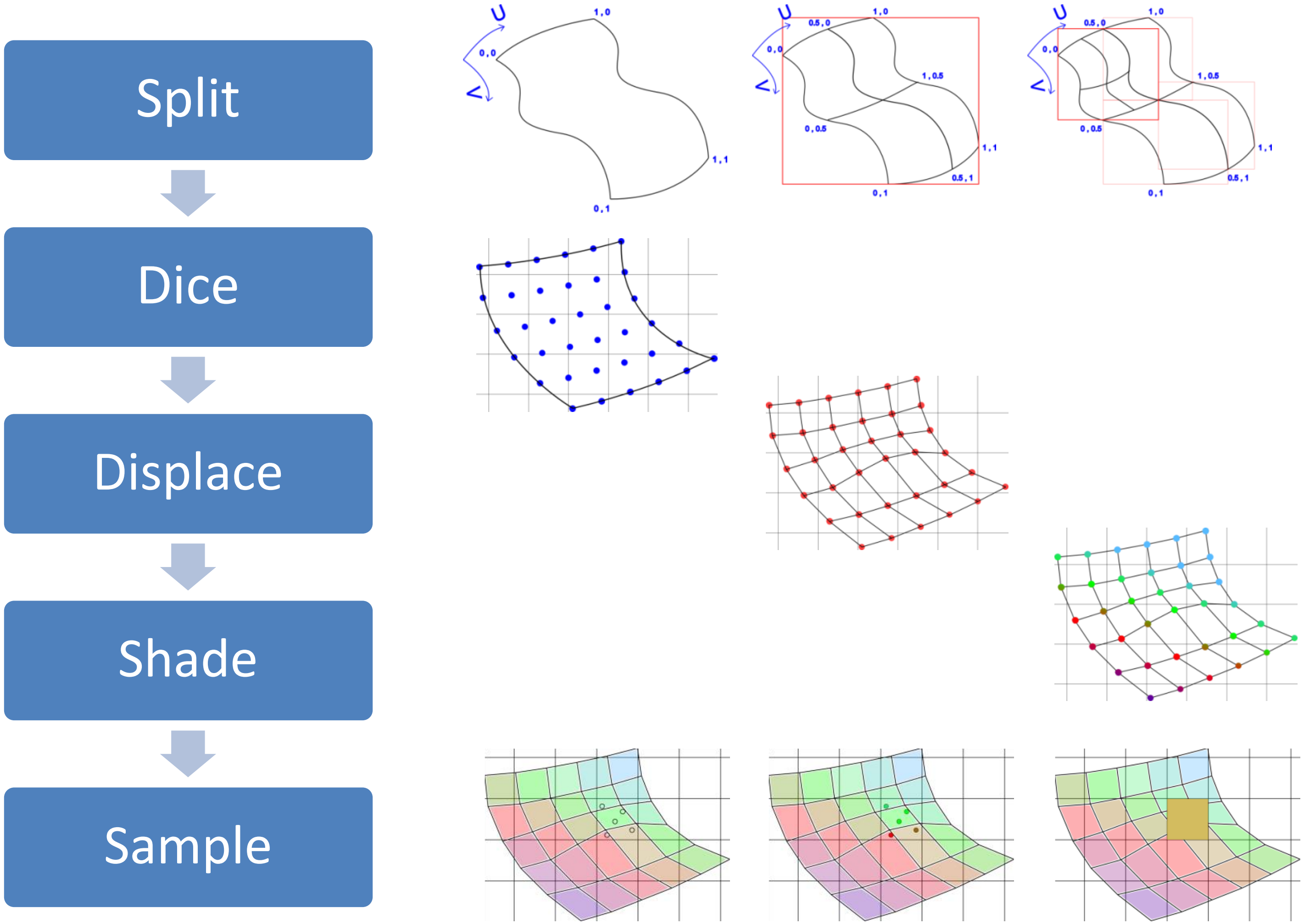
RiSphere

RiParaboloid

REYES features

- Native support for high level surfaces
- Dynamic LOD
  - Compact representation
  - Subdivide per-frame based on size on screen
  - Displace geometry from textures
- High quality filtering
- Easier to deal with translucency, motion-blur, etc.
- Can be used together with ray-tracing

REYES pipeline overview



Split

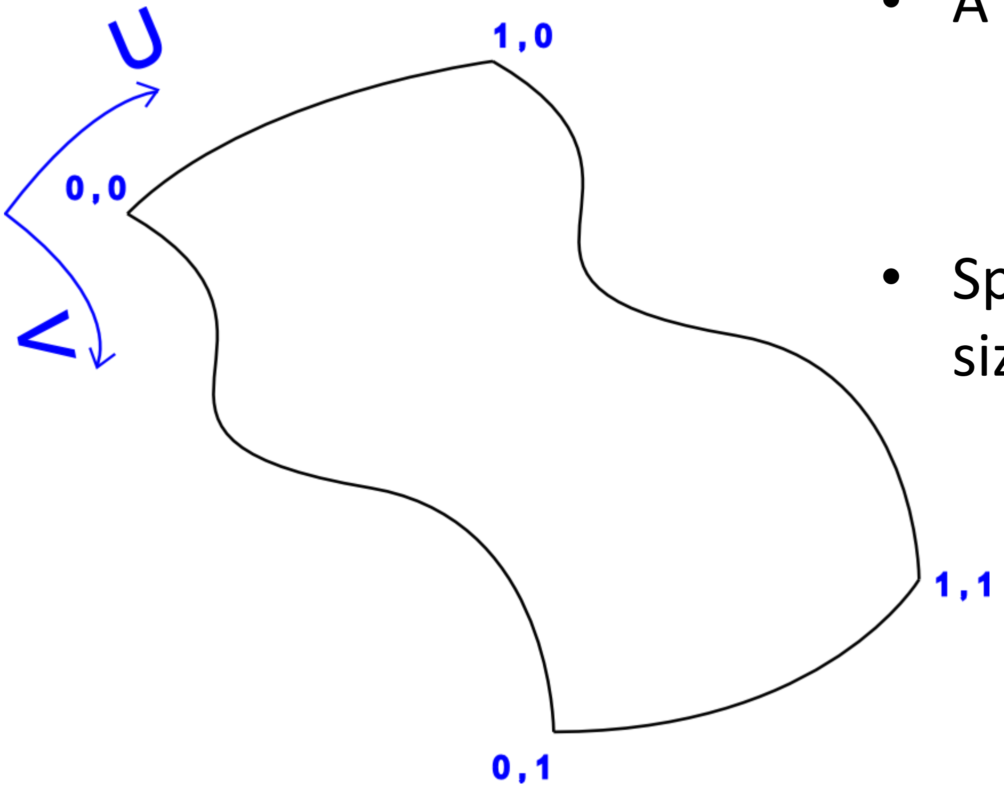
Dice

Displace

Shade

Sample

Split (1)

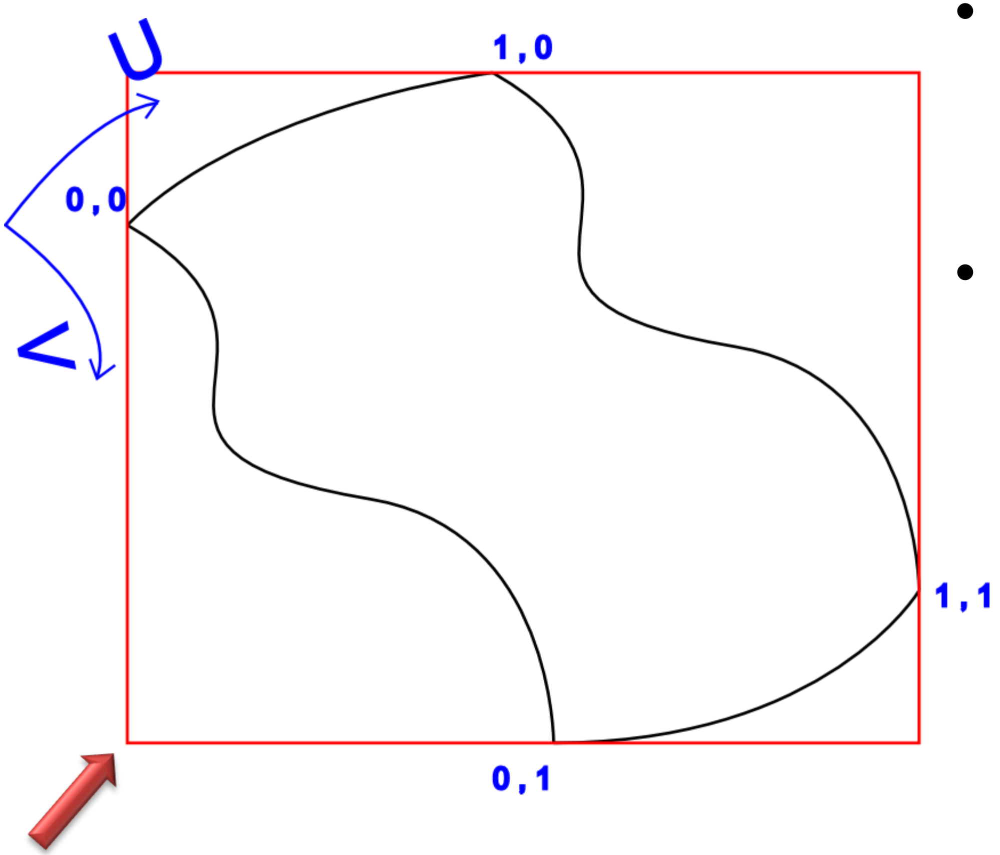


- A parametric surface:

  P = f(u,v)

- Split until "small enough" size (estimated) on screen

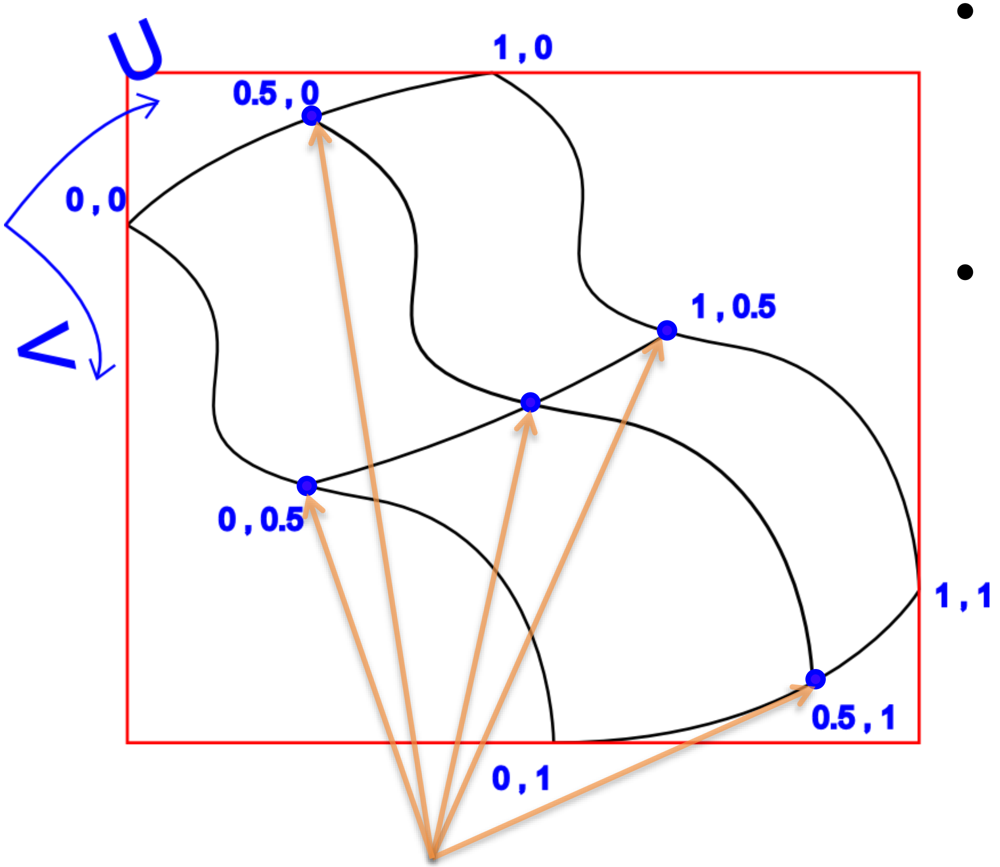
Split (2)



Here, the bounding box is too large..

- Calculate the bounding box in *screen-space*
- …test against predetermined max screen area

Split (3)



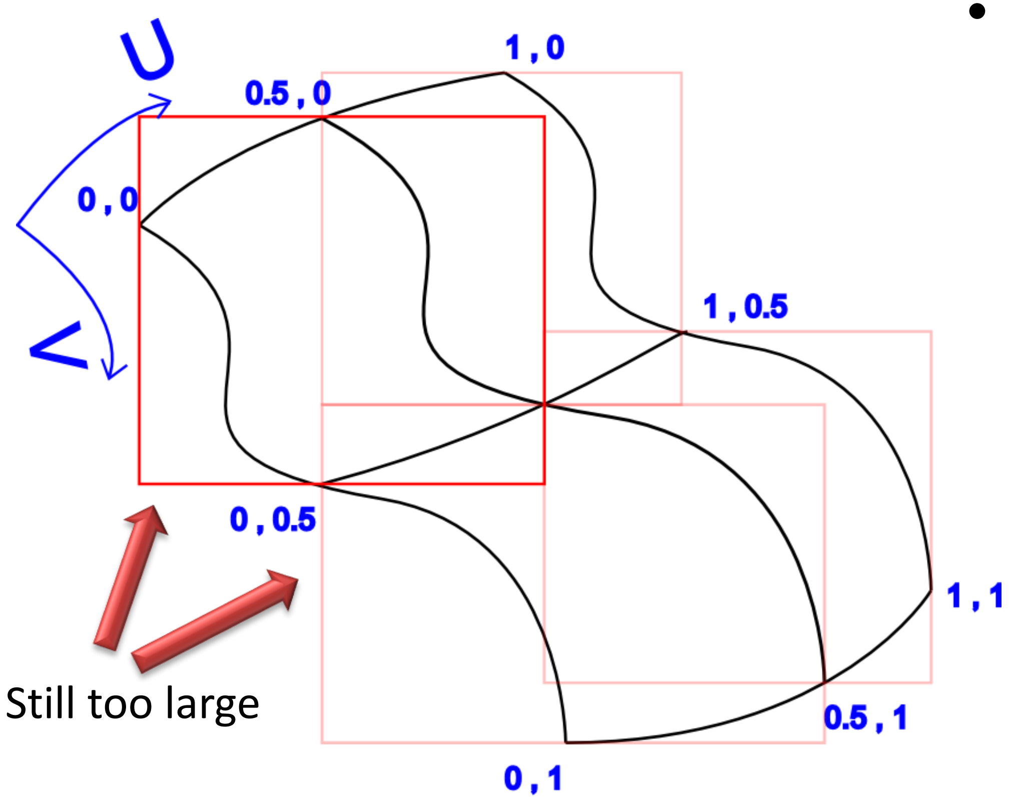
New split points

- When too large, **split** the patch

- It's easy with parametric primitives:

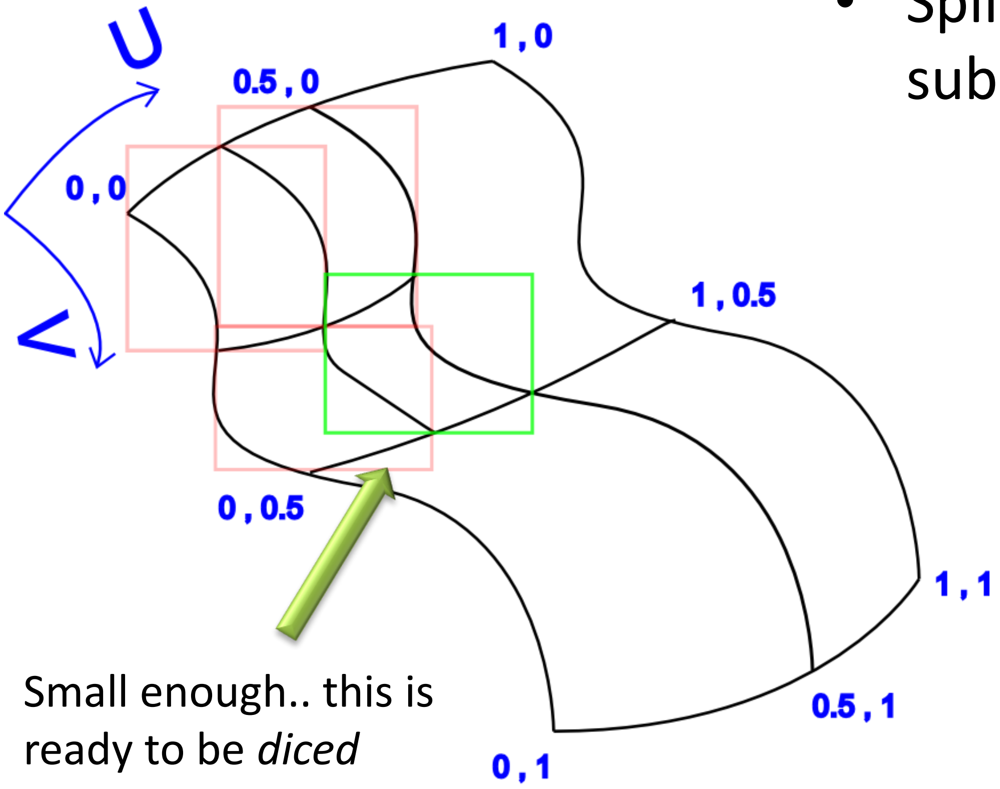
  $P_{new} = f(u_{new}, v_{new})$

Split (4)



- Calculate the bounds of the new sub-patches

Split (5)



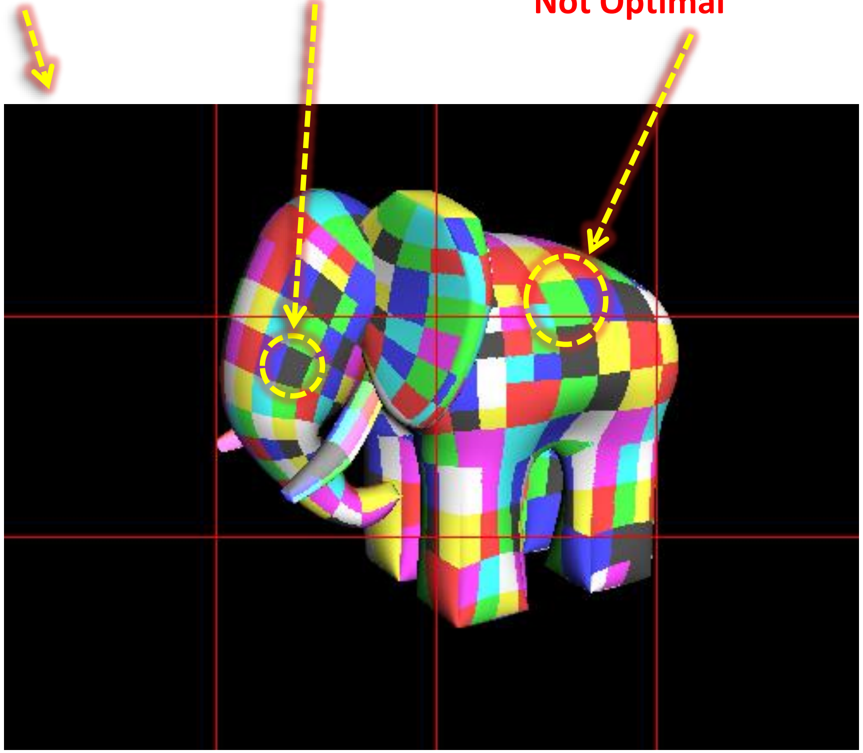
Small enough.. this is ready to be *diced*

- Split recursively until every sub-patch is “small enough”…

What’s “Small Enough” ?

- When most sub-patches fit in a single *bucket*

- When *dicing* (see later) produces a suitable number of samples (sweet spot for performance)

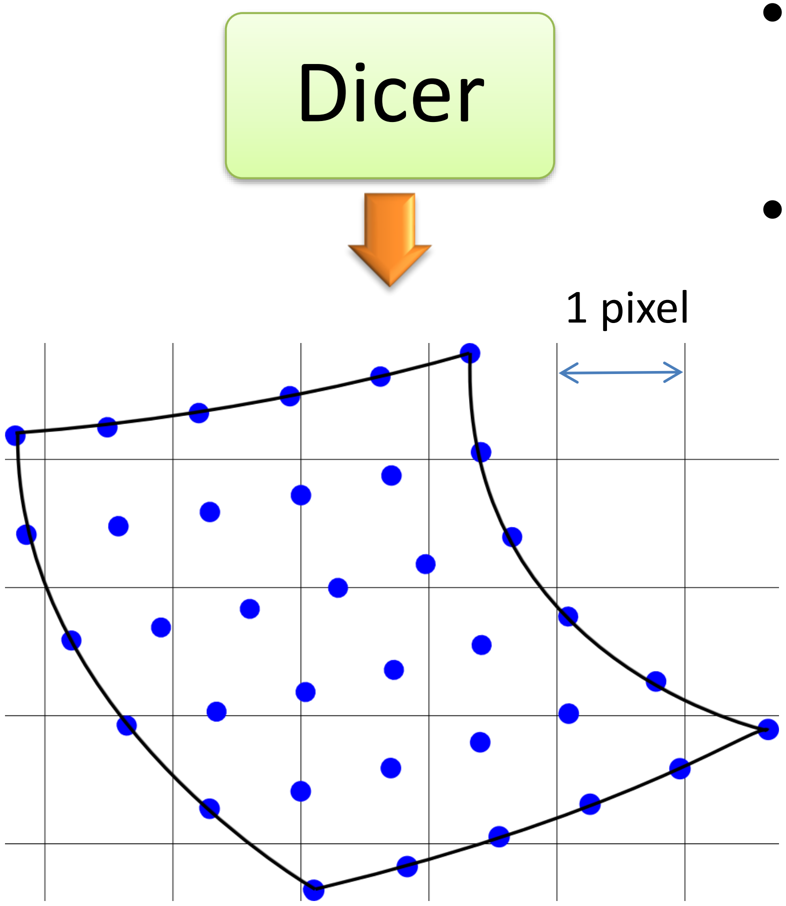


A bucket

Sub-patch inside a single bucket
Optimal

Sub-patch touches two or more buckets
Not Optimal

Dice


Dicer

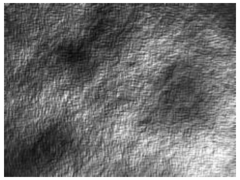
- Small enough sub-patches are **diced**
- Generate a dense **grid** of samples (1 pixel-per-sample or more..)

Small enough ← n_samples <= max_samples

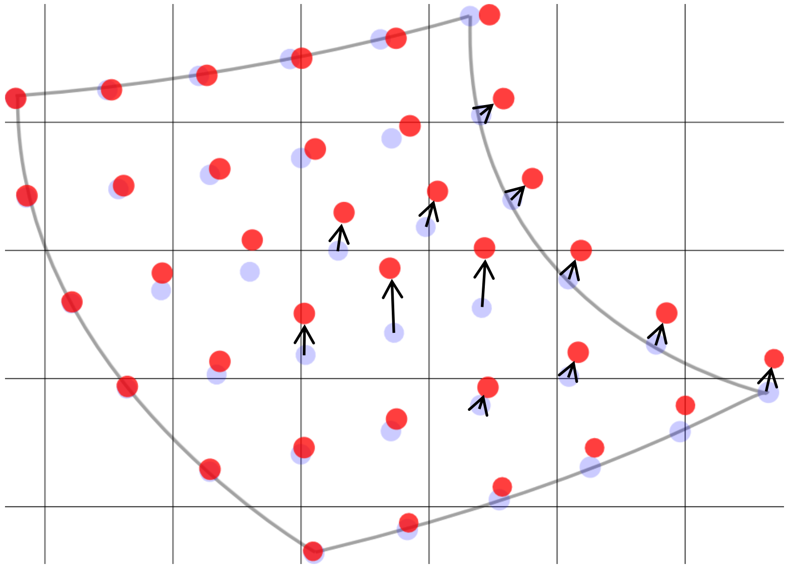
max_samples is set for performance reasons and to avoid distortion

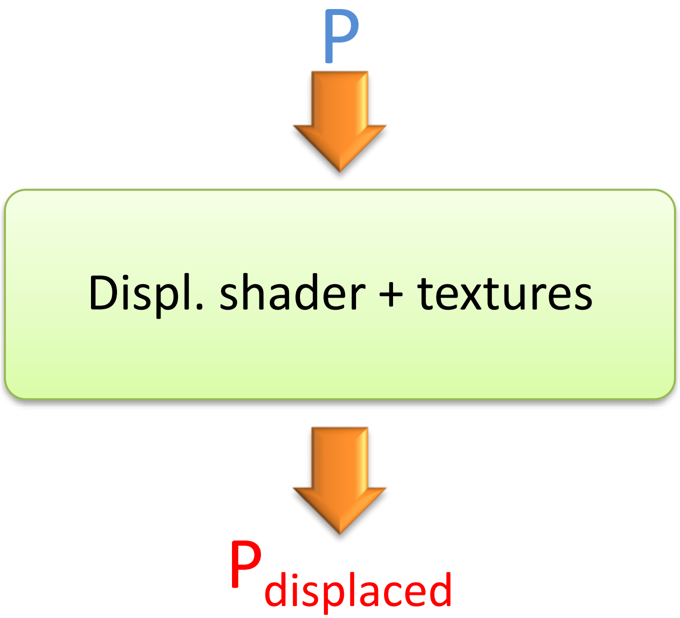
Displace

```
myDisplace()
{
   mag = texture( “dispmap” );
   P += normalize(N) * mag;
   N = calculatenormal(P);
}
```



Apply a displacement shader to the position of the samples

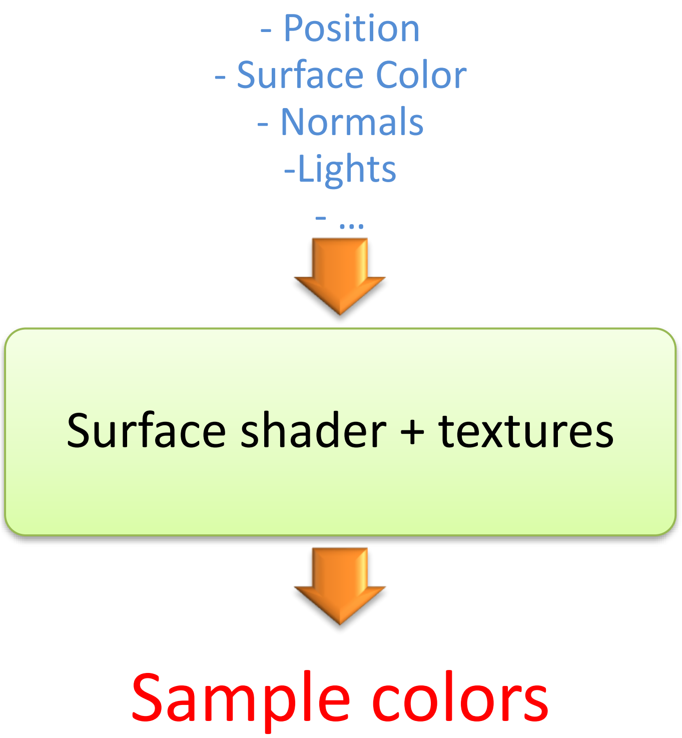




Shade

```
myShader()
{
   txcol = texture( “pigment” );
   Ci = diffuse(N,txcol);
   Oi = 1;
}
```

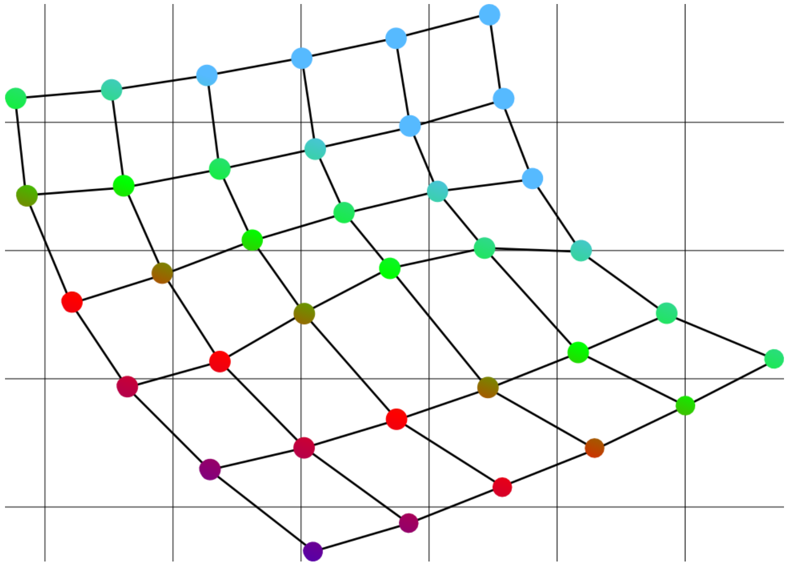
Apply surface and light shaders to get the color



- Position
- Surface Color
- Normals
-Lights
- …

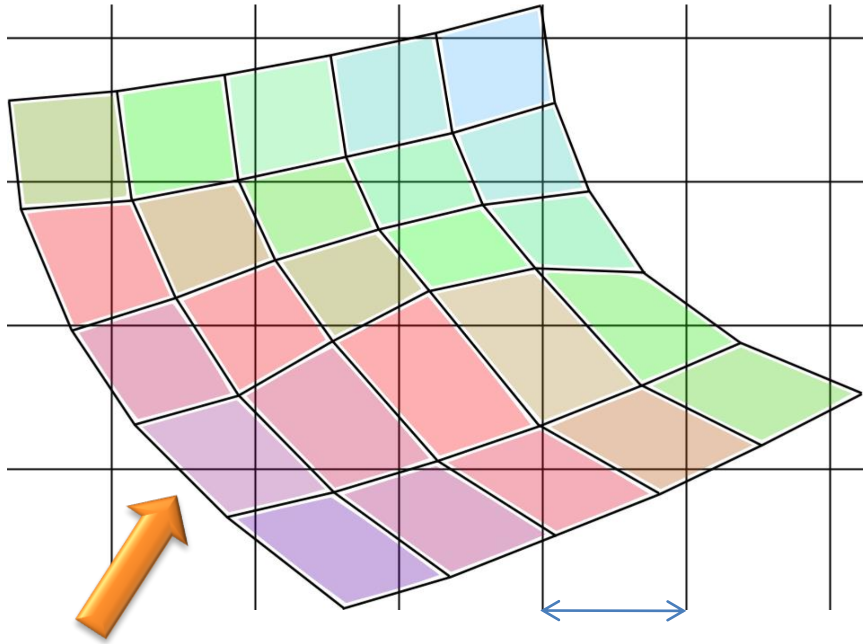
Surface shader + textures

Sample colors



Sample – the micropolys

Form *virtual* micropolygons at the grid samples

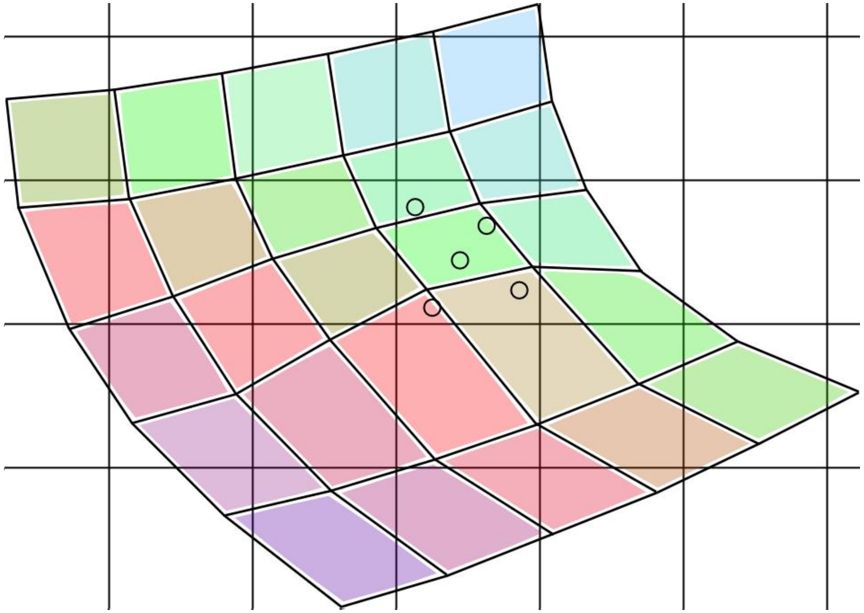


Virtual micropolygons

1 pixel

Sample – sample points

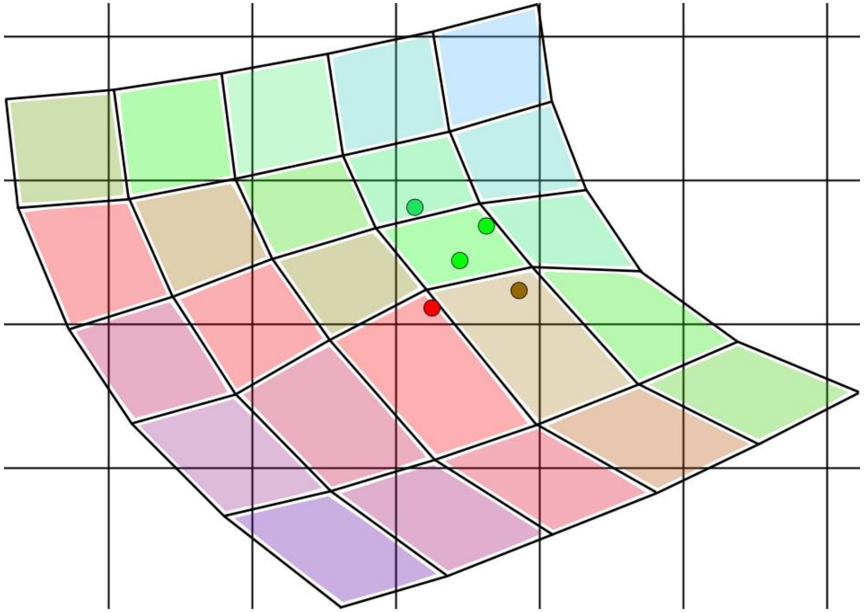
Multiple sub-samples at every pixel



…choose a sampling method: regular, multi-jittered (as shown), etc.

Sample – gather samples

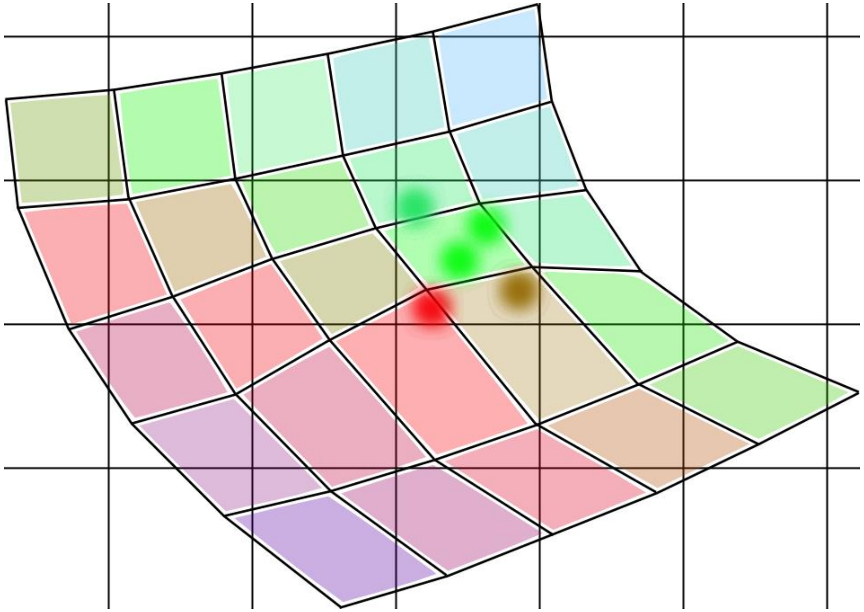
Samples get the color of the micropolygons they touch



…each sample can have many values if the mpolys are translucent !

Sample – convolution

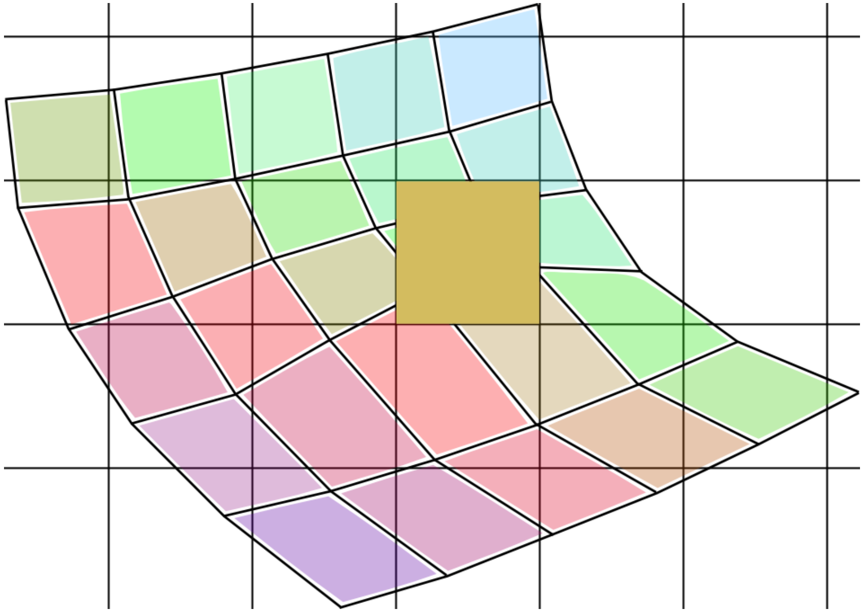
Mix the samples together…



…choose a filter: box, triangular, Gaussian, Sinc, etc..

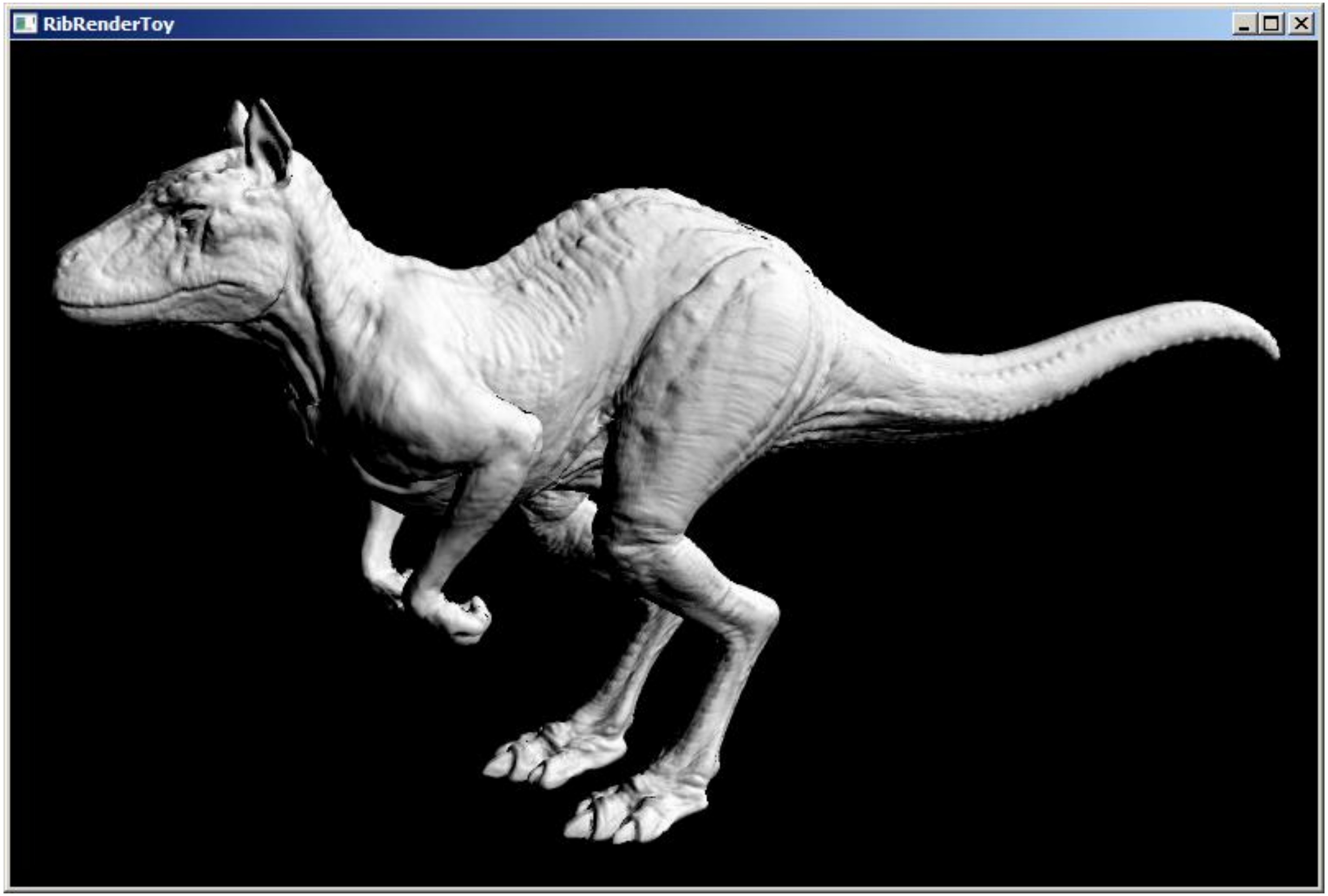
Sample – final pixel color

The resulting "average" color is assigned to the pixel



…repeat for every pixel 8)

RibTools: A RenderMan-style renderer R&D



* Killeroo model from headus Ltd., used with permission.

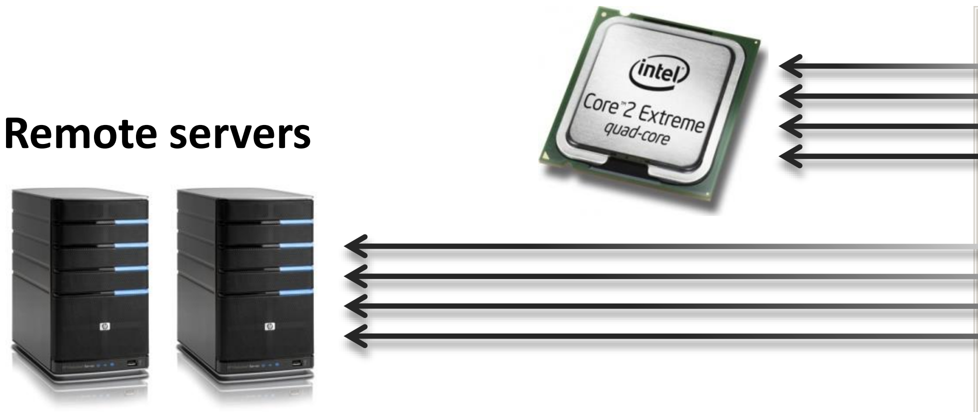
RibTools’ key features

PIXAR's RenderMan

- RenderMan compliant (…almost (^^;))
  - Parse RIB scene files
  - C-like shaders compiler and VM
  - Parametric surfaces, etc.
  - Sub-pixel displacement mapping
- Open Sourced (BSD License)
- Multi-threaded
- Network-distributed
- “Future proof” SIMD

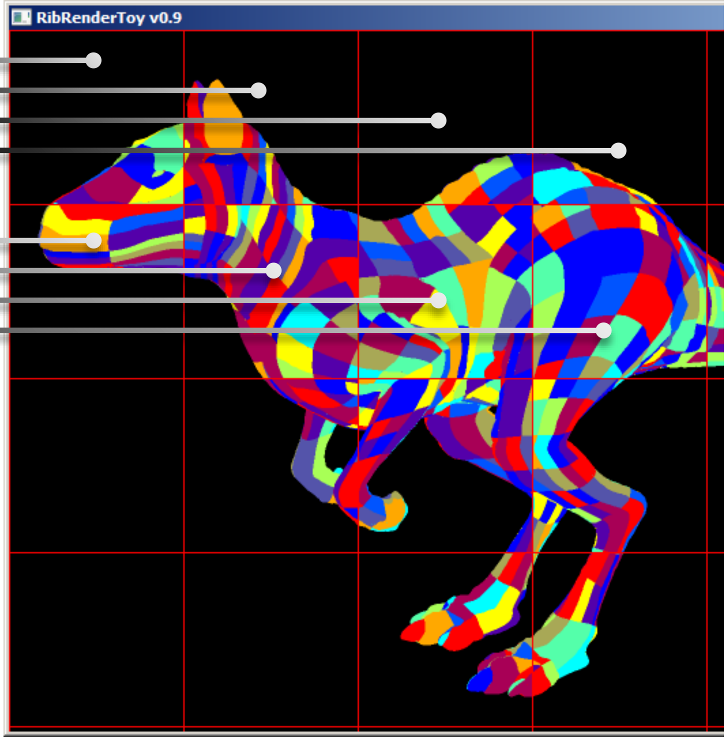
Scalability !

Distributed bucket rendering



Multi-core CPU

Remote servers

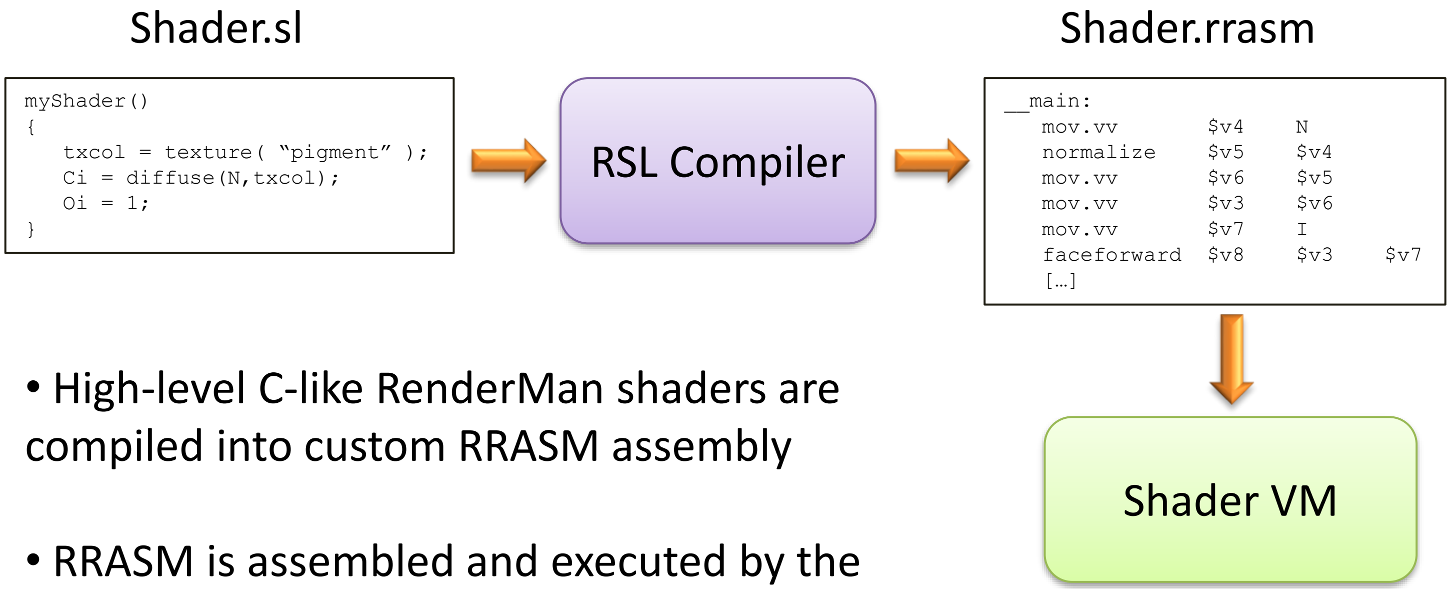


- A frame is subdivided into discrete **buckets**
- Buckets are assigned to **threads** on the CPU or to remote servers via **TCP/IP**
- Geometry, shaders and textures are also transferred via TCP/IP

It works today .... …it’s only a start. It needs optimizations, esp. network.

Shader system

A shading system is an essential part of a renderer



Shader.sl

```
myShader()
{
   txcol = texture( “pigment” );
   Ci = diffuse(N,txcol);
   Oi = 1;
}
```

Shader.rrasm

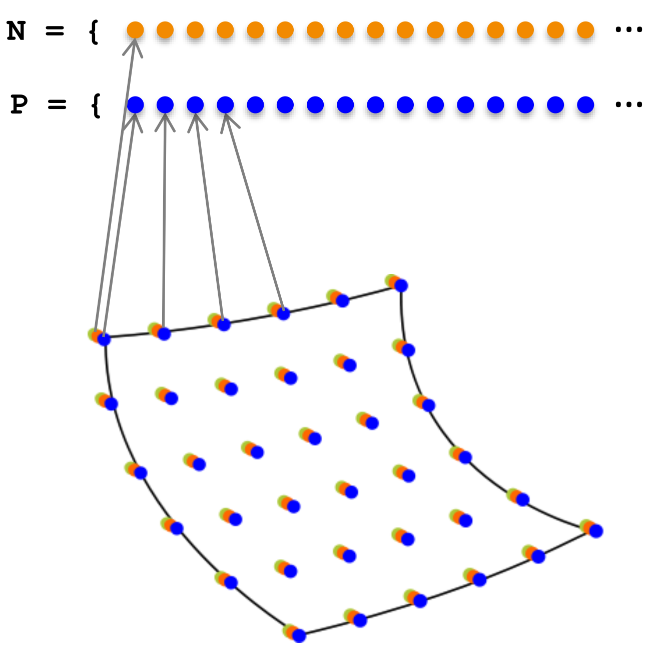
```
__main:
   mov.vv       $v4     N
   normalize    $v5     $v4
   mov.vv       $v6     $v5
   mov.vv       $v3     $v6
   mov.vv       $v7     I
   faceforward  $v8     $v3     $v7
   […]
```

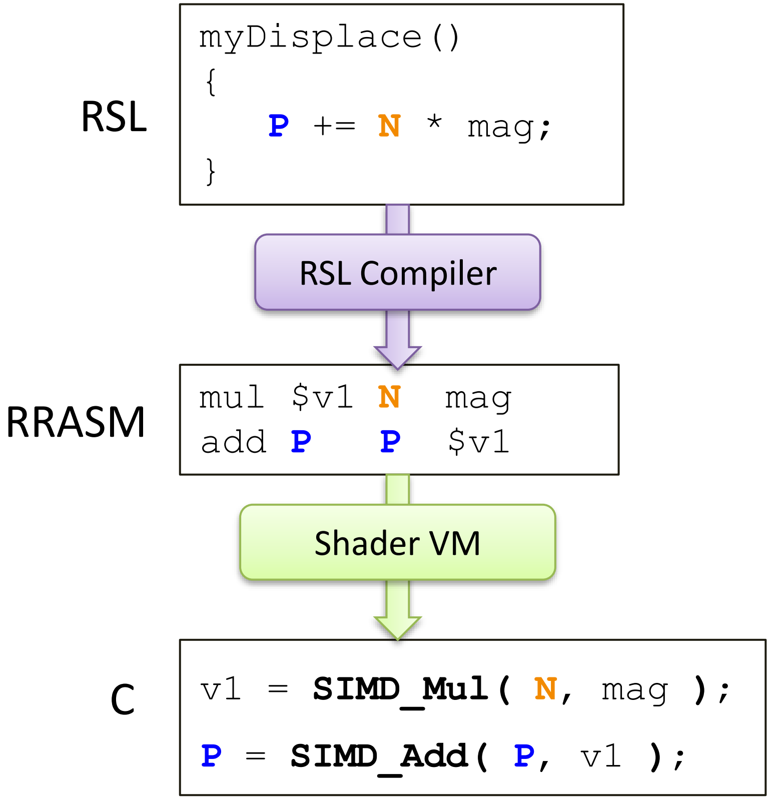
- High-level C-like RenderMan shaders are compiled into custom RRASM assembly

- RRASM is assembled and executed by the Shader **Virtual Machine (VM)** when rendering

Shading and SIMD (1)

Values in a grid are treated as arrays…





RSL

```
myDisplace()
{
    P += N * mag;
}
```

RSL Compiler

RRASM

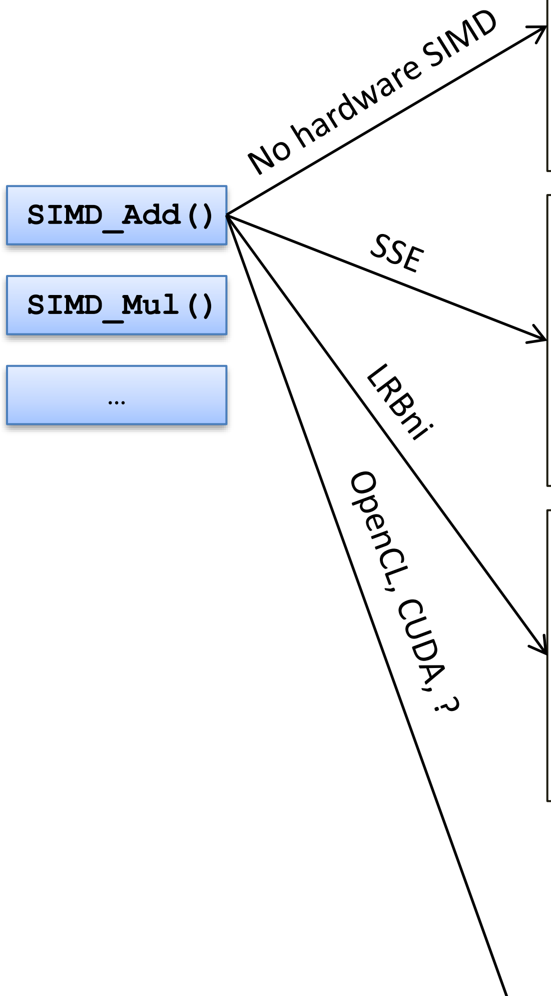
```
mul $v1 N  mag
add P   P  $v1
```

Shader VM

C

```
v1 = SIMD_Mul( N, mag );
P = SIMD_Add( P, v1 );
```

Shading and SIMD (2)



```
Vector SIMD_Add_FPU( Vector a, Vector b )
{
   for (i=0; i < vec_size; i += 1)
   {
      result[i] = a[i] + b[i];
   }
   return result;
}
```

1x

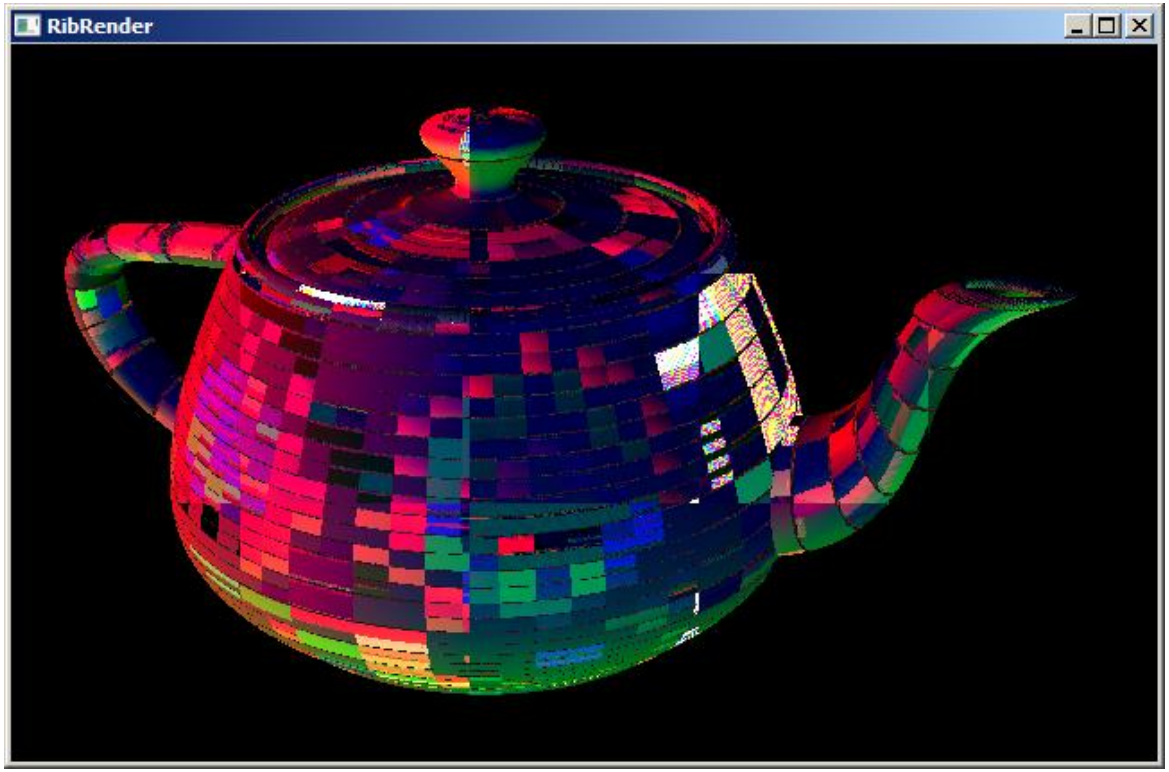
```
Vector SIMD_Add_SSE( Vector a, Vector b )
{
   for (i=0; i < vec_size; i += 4)
   {
      SSE_Add( &result[i], &a[i], &b[i] );
   }
   return result;
}
```

4x

```
Vector SIMD_Add_LRB( Vector a, Vector b )
{
   for (i=0; i < vec_size; i += 16)
   {
      LRB_Add( &result[i], &a[i], &b[i] );
   }
   return result;
}
```

16x

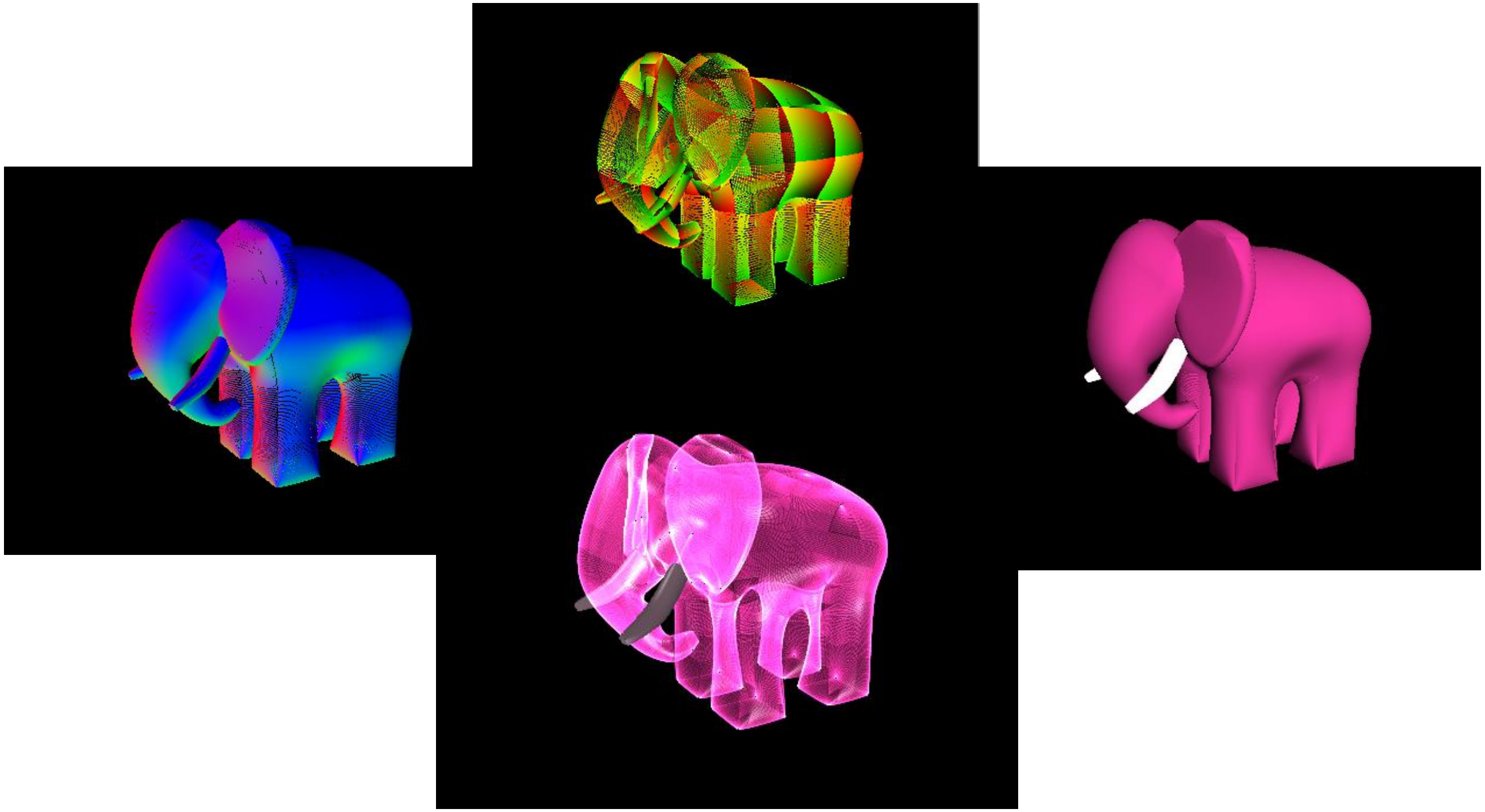
…**not** fun to debug !



Cons and problems

- Requires highly programmable hardware (best if with a flexible texture unit)
- The “RenderMan interface” is a fairly deep standard to follow
- Shader compilers, optimizers.. complex stuff
- Comes with other issues:
  - Cracks when tessellating, non-planar micropolys, front plane clipping, etc.

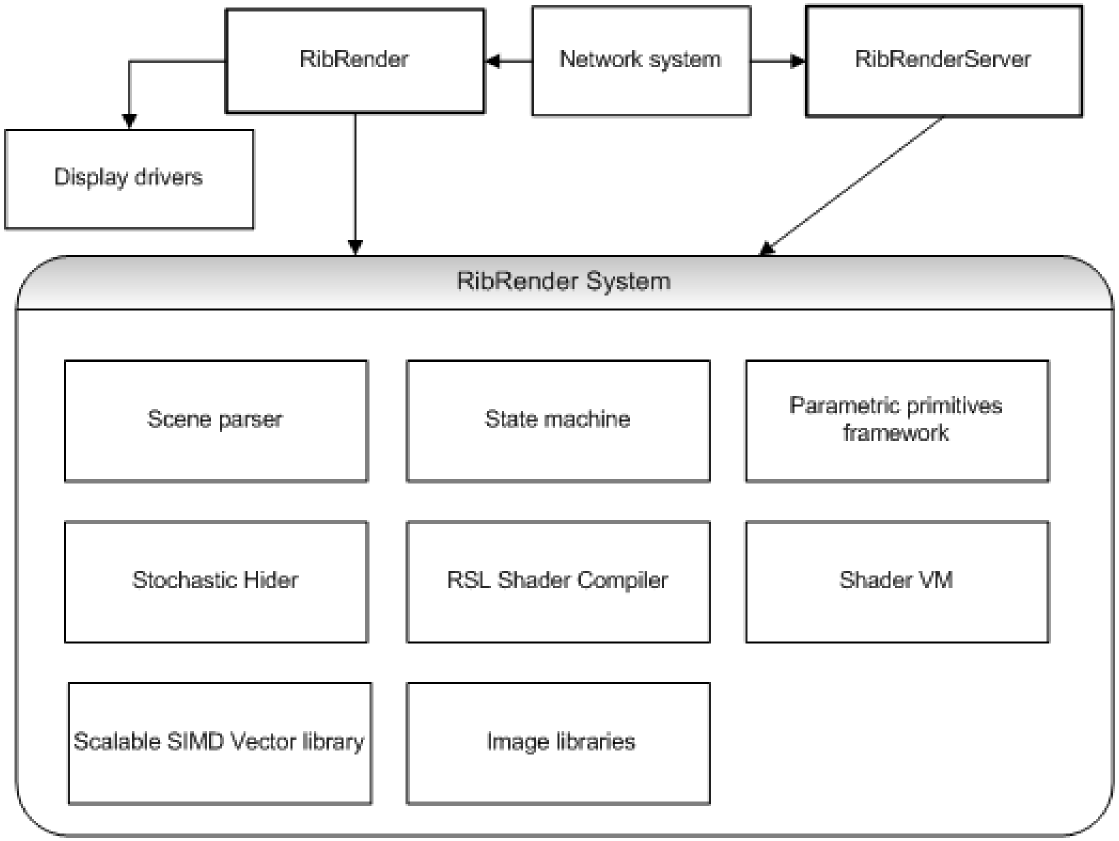
Questions ?



References

- RibTools source code on GitHub
  - https://github.com/dpasca/RibTools
- “The RenderMan Interface Specification” (aka RISpec)
  - https://renderman.pixar.com/products/rispec/
- “Rendering with REYES” (from Pixar)
  - https://renderman.pixar.com/products/whats_renderman/2.html
- “Production Rendering” (Ian Stephenson Ed.)
  - http://amazon.com/dp/1852338210
- “Advanced RenderMan” (by A.Apodaca and L.Gritz)
  - http://amazon.com/dp/1558606181
- “The RenderMan Companion” (by Steve Upstill)
  - http://amazon.com/dp/0201508680

Appendix: RibTools system overview


RibRender
Network system
RibRenderServer
Display drivers
RibRender System
Scene parser
State machine
Parametric primitives framework
Stochastic Hider
RSL Shader Compiler
Shader VM
Scalable SIMD Vector library
Image libraries