GPU Shading and Rendering: Introduction & Graphics Hardware

Marc Olano

Computer Science and Electrical Engineering
University of Maryland, Baltimore County

SIGGRAPH 2005
<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
<th>Presenter(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30</td>
<td>Intro/Hardware</td>
<td>Olano</td>
</tr>
<tr>
<td>9:25</td>
<td>Compilers</td>
<td>Bleiweiss</td>
</tr>
<tr>
<td>10:30</td>
<td>Shading Languages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GLSL</td>
<td>Olano</td>
</tr>
<tr>
<td>10:55</td>
<td>Cg</td>
<td>Kilgard</td>
</tr>
<tr>
<td>11:20</td>
<td>HLSL</td>
<td>Sander</td>
</tr>
<tr>
<td>11:45</td>
<td>Sh</td>
<td>McCool</td>
</tr>
<tr>
<td>1:45</td>
<td>Rendering Algorithms</td>
<td>Hart</td>
</tr>
<tr>
<td>2:35</td>
<td>GPU Production</td>
<td>Gritz</td>
</tr>
<tr>
<td></td>
<td>Rendering Algorithms</td>
<td></td>
</tr>
<tr>
<td>3:45</td>
<td>ATI</td>
<td>Sander</td>
</tr>
<tr>
<td>4:25</td>
<td>NVIDIA</td>
<td>Kilgard</td>
</tr>
<tr>
<td>5:05</td>
<td>Panel Q&amp;A</td>
<td>All</td>
</tr>
</tbody>
</table>
Part I

Introduction
What is a GPU?

- Graphics Processing Unit
  - Graphics accelerator
  - Parallel processing unit
- We’re doing graphics, what is it good for?
  - Better real-time graphics
  - Faster non-real-time graphics
What is a GPU?

- Graphics Processing Unit
  - Graphics accelerator
  - Parallel processing unit
- We’re doing graphics, what is it good for?
  - Better real-time graphics
  - Faster non-real-time graphics
What is a GPU?

- Graphics Processing Unit
  - Graphics accelerator
  - Parallel processing unit
- We’re doing graphics, what is it good for?
  - Better real-time graphics
  - Faster non-real-time graphics
What is a GPU?

- Graphics Processing Unit
  - Graphics accelerator
  - Parallel processing unit
- We’re doing graphics, what is it good for?
  - Better real-time graphics
  - Faster non-real-time graphics
What is Shading?

- What color are the pixels
- Programmable
  - Flexible Appearance
  - Arbitrary computation
- Procedural

Simple procedures
High-level language
What is Shading?

• What color are the pixels
• Programmable
  • Flexible Appearance
  • Arbitrary computation
• Procedural
What is Shading?

- What color are the pixels
- Programmable
  - Flexible Appearance
  - Arbitrary computation
- Procedural
  - Simple procedures
What is Shading?

- What color are the pixels
- Programmable
  - Flexible Appearance
  - Arbitrary computation
- Procedural
  - Simple procedures
  - High-level language
What is Shading?

- What color are the pixels
- Programmable
  - Flexible Appearance
  - Arbitrary computation
- Procedural
  - Simple procedures
  - High-level language
What is Shading?

- What color are the pixels
- Programmable
  - Flexible Appearance
  - Arbitrary computation
- Procedural
  - Simple procedures
  - High-level language
What is Shading?

- What color are the pixels
- Programmable
  - Flexible Appearance
  - Arbitrary computation
- Procedural
  - Simple procedures
  - High-level language
Some examples

- More realistic appearance
  - Bump mapping, Anisotropic, Precomputed radiance transfer, ...
- Non-realistic appearance
  - Cartoon, Sketch, Illustration, ...
- Animated appearance
  - Skinning, Water, Clouds, ...
- Visualization
  - Data on surfaces, Volume rendering, ...
Some examples

• More realistic appearance
  • Bump mapping, Anisotropic, Precomputed radiance transfer, ...

• Non-realistic appearance
  • Cartoon, Sketch, Illustration, ...

• Animated appearance
  • Skinning, Water, Clouds, ...

• Visualization
  • Data on surfaces, Volume rendering, ...
Some examples

- More realistic appearance
  - Bump mapping, Anisotropic, Precomputed radiance transfer, ...

- Non-realistic appearance
  - Cartoon, Sketch, Illustration, ...

- Animated appearance
  - Skinning, Water, Clouds, ...

- Visualization
  - Data on surfaces, Volume rendering, ...
Some examples

- More realistic appearance
  - Bump mapping, Anisotropic, Precomputed radiance transfer, ...
- Non-realistic appearance
  - Cartoon, Sketch, Illustration, ...
- Animated appearance
  - Skinning, Water, Clouds, ...
- Visualization
  - Data on surfaces, Volume rendering, ...
What is Rendering?

• The rest of the problem!
• In our case, using GPU for other than polygon rendering
  • Curved surfaces
  • Ray tracing
  • Point based rendering
  • ...
What is Rendering?

- The rest of the problem!
- In our case, using GPU for other than polygon rendering
  - Curved surfaces
  - Ray tracing
  - Point based rendering
  - ...
What is Rendering?

- The rest of the problem!
- In our case, using GPU for other than polygon rendering
  - Curved surfaces
  - Ray tracing
  - Point based rendering
  - ...
<table>
<thead>
<tr>
<th>Non-Real Time vs. Real-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real-Time</strong></td>
</tr>
<tr>
<td>Tens of frames per second</td>
</tr>
<tr>
<td>Thousand instruction shaders</td>
</tr>
<tr>
<td>Limited computation, texture, memory, ...</td>
</tr>
<tr>
<td><strong>Non-Real-Time</strong></td>
</tr>
<tr>
<td>Seconds to hours per frame</td>
</tr>
<tr>
<td>Thousands line shaders</td>
</tr>
<tr>
<td>“Unlimited” computation, texture, memory, ...</td>
</tr>
</tbody>
</table>
## Non-Real Time vs. Real-Time

<table>
<thead>
<tr>
<th>Real-Time</th>
<th>Non-Real-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tens of frames per second</td>
<td>Seconds to hours per frame</td>
</tr>
<tr>
<td>Thousand instruction shaders</td>
<td>Thousands line shaders</td>
</tr>
<tr>
<td>Limited computation, texture, memory, ...</td>
<td>“Unlimited” computation, texture, memory, ...</td>
</tr>
</tbody>
</table>
## Non-Real Time vs. Real-Time

<table>
<thead>
<tr>
<th>Real-Time</th>
<th>Non-Real-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tens of frames per second</td>
<td>Seconds to hours per frame</td>
</tr>
<tr>
<td>Thousand instruction shaders</td>
<td>Thousands line shaders</td>
</tr>
<tr>
<td>Limited computation, texture, memory, ...</td>
<td>“Unlimited” computation, texture, memory, ...</td>
</tr>
</tbody>
</table>
## Non-Real Time vs. Real-Time

<table>
<thead>
<tr>
<th>Real-Time</th>
<th>Non-Real-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tens of frames per second</td>
<td>Seconds to hours per frame</td>
</tr>
<tr>
<td>Thousand instruction shaders</td>
<td>Thousands line shaders</td>
</tr>
<tr>
<td>Limited computation, texture,</td>
<td>“Unlimited” computation, texture, memory,</td>
</tr>
<tr>
<td>memory, ...</td>
<td>...</td>
</tr>
</tbody>
</table>
How is this possible?

- GPUs are programmable!
  - Per-vertex programs
  - Per-fragment programs
Research Languages

- Pixel-Planes 5 [Rhoades et al., 1992]
- PixelFlow/pfman [Olano and Lastra, 1998]
- RTSL [Proudfoot et al., 2001]
- Sh [McCool and Toit, 2004]
- (BrookGPU) [Buck et al., 2004]
Research Languages

- Pixel-Planes 5 [Rhoades et al., 1992]
- PixelFlow/pfman [Olano and Lastra, 1998]
- RTSL [Proudfoot et al., 2001]
- Sh [McCool and Toit, 2004]
- (BrookGPU) [Buck et al., 2004]
Research Languages

- Pixel-Planes 5 [Rhoades et al., 1992]
- PixelFlow/pfman [Olano and Lastra, 1998]
- RTSL [Proudfoot et al., 2001]
  - Sh [McCool and Toit, 2004]
  - (BrookGPU) [Buck et al., 2004]
Research Languages

- Pixel-Planes 5 [Rhoades et al., 1992]
- PixelFlow/pfman [Olano and Lastra, 1998]
- RTSL [Proudfoot et al., 2001]
- Sh [McCool and Toit, 2004]
- (BrookGPU) [Buck et al., 2004]
Research Languages

- Pixel-Planes 5 [Rhoades et al., 1992]
- PixelFlow/pfman [Olano and Lastra, 1998]
- RTSL [Proudfoot et al., 2001]
- Sh [McCool and Toit, 2004]
- (BrookGPU) [Buck et al., 2004]
Commercial Languages

- GL or DX low-level
- OpenGL Shading Language
- DirectX HLSL
- NVIDIA Cg
Commercial Languages

- GL or DX low-level
- OpenGL Shading Language
  - DirectX HLSL
  - NVIDIA Cg
Commercial Languages

- GL or DX low-level
- OpenGL Shading Language
- DirectX HLSL
- NVIDIA Cg
Commercial Languages

- GL or DX low-level
- OpenGL Shading Language
- DirectX HLSL
- NVIDIA Cg
Part II

Graphics Hardware
Ignoring Hardware Differences

Outline

Ignoring Hardware Differences
Simplified Models
RenderMan
Hardware
Machine Complexity

- Graphics machines are complex
- User does not want to know
  - How machine does what it does
  - Tons of machine-specific differences
- Answer:
  - Simple model of machine
Ignoring Hardware Differences

Machine Complexity

- Graphics machines are complex
- User does not want to know
  - How machine does what it does
  - Tons of machine-specific differences

Answer:
- Simple model of machine
- High-level language for procedures
- Well-defined procedure input & output
- System connects procedures
Machine Complexity

- Graphics machines are complex
- User does not want to know
  - How machine does what it does
  - Tons of machine-specific differences
- Answer:
  - Simple model of machine
  - High-level language for procedures
  - Well-defined procedure input & output
  - System connects procedures
Graphics machines are complex
User does not want to know
  - How machine does what it does
  - Tons of machine-specific differences
Answer:
  - Simple model of machine
    - High-level language for procedures
    - Well-defined procedure input & output
    - System connects procedures
Ignoring Hardware Differences

Machine Complexity

- Graphics machines are complex
- User does not want to know
  - How machine does what it does
  - Tons of machine-specific differences
- Answer:
  - Simple model of machine
  - High-level language for procedures
    - Well-defined procedure input & output
    - System connects procedures
Machine Complexity

- Graphics machines are complex
- User does not want to know
  - How machine does what it does
  - Tons of machine-specific differences
- Answer:
  - Simple model of machine
  - High-level language for procedures
  - Well-defined procedure input & output
  - System connects procedures
Machine Complexity

- Graphics machines are complex
- User does not want to know
  - How machine does what it does
  - Tons of machine-specific differences
- Answer:
  - Simple model of machine
  - High-level language for procedures
  - Well-defined procedure input & output
  - System connects procedures
Simplified Machine

- User’s mental model
- Hide details
- Device independent
- Procedural stages
Simplified Machine

- User’s mental model
- Hide details
  - Device independent
  - Procedural stages
Simplified Machine

- User’s mental model
- Hide details
- Device independent
  - Procedural stages
Simplified Machine

- User’s mental model
- Hide details
- Device independent
- Procedural stages
Ignoring Hardware Differences

RenderMan Model

- "Abstract" interface
  - Blocks = procedures
  - Block interfaces well defined

- Connections
  - Inputs & outputs don’t have to match
  - System handles conversion
Ignoring Hardware Differences

RenderMan Model

- “Abstract” interface
  - Blocks = procedures
  - Block interfaces well defined
- Connections
  - Inputs & outputs don’t have to match
  - System handles conversion
RenderMan
Shader types

Model
Transform
Displacement
Shade
Light
Volume
Imager

Ignoring Hardware Differences
Ignoring Hardware Differences

RenderMan
Shader types

Model
Transform
Displacement
Shade
Light
Volume
Imager
Ignoring Hardware Differences

RenderMan
Shader types

Model
Transform
Displacement
Shade
Light
Volume
Imager
RenderMan
Shader types

Model
Transform
Displacement
Shade
Light
Volume
Imager
Ignoring Hardware Differences

RenderMan
Shader types

Model → Transform → Displacement → Shade → Light → Volume → Imager
RenderMan
Shader types

Model
Transform
Displacement
Shade
Light
Volume
Imager
RenderMan Model

- What it says:
  - Input and output of each block
  - What each block should do

- What it doesn’t say:
  - Order or grouping of processing
Ignoring Hardware Differences

RenderMan Model

- What it says:
  - Input and output of each block
  - What each block should do

- What it doesn’t say:
  - Order or grouping of processing
Ignoring Hardware Differences

RenderMan
Ray Tracing

- Model
- Transform
- Displacement
- Shade
- Light
- Volume
- Imager

Ray

Image / Pixel
RenderMan
SGI Multi-pass RenderMan

Model → Transform → Displacement → Shade → Light → Volume → Imager

Object / Pass / Pixel

Image / Pixel

Object
Ignoring Hardware Differences

Hardware Model

- Vertex shading
  - Transform
    - Procedural transformation
    - Skinning
  - Shade
    - Per-vertex shading
    - Computed texture coordinates
Hardware Model

- Fragment shading
  - Per-fragment shading
  - Computed and dependent texture
Ignoring Hardware Differences

Hardware Model

- Render to texture
  - Rendered shadow & environment maps
  - Multi-pass fragment shading

[Proudfoot et al., 2001]
Ignoring Hardware Differences

Hardware Model

- Render to vertex array / buffer objects
  - Geometry images [Gu et al., 2002]
  - Multi-pass vertex shading
  - Merge vertex & fragment capabilities
Hardware Model

- Vertex texture
  - Texture-based vertex displacement
  - Tabulated functions
Ignoring Hardware Differences

Hardware Model

- It's all about the memory
- What it says:
  - Input and output of each block
  - What each block should do
- What it doesn’t say:
  - Vertex processing order
  - Fragment processing order
  - Interleaving of vertex and fragment
Ignoring Hardware Differences

Hardware Model

- It's all about the memory
- What it says:
  - Input and output of each block
  - What each block should do
- What it doesn’t say:
  - Vertex processing order
  - Fragment processing order
  - Interleaving of vertex and fragment
Ignoring Hardware Differences

Hardware Model

- It's all about the memory
- What it says:
  - Input and output of each block
  - What each block should do
- What it doesn’t say:
  - Vertex processing order
  - Fragment processing order
  - Interleaving of vertex and fragment
No, but really, what’s in there?

- Some other stuff,
- Parallelism,
- And more parallelism
No, but really, what’s in there?

- Some other stuff,
- Parallelism,
- And more parallelism
No, but really, what’s in there?

- Some other stuff,
- Parallelism,
- And more parallelism
No, but really, what’s in there?

- Some other stuff,
- Parallelism,
- And more parallelism
Part III

Noise
Outline

What is this Noise?

Perlin noise

Modifications
Why Noise?

- Introduced by [Perlin, 1985]
  - Heavily used in production animation
  - Technical Achievement Oscar in 1997
  - “Salt,” adds spice to shaders
Why Noise?

- Introduced by [Perlin, 1985]
  - Heavily used in production animation
  - Technical Achievement Oscar in 1997
- “Salt,” adds spice to shaders
What is this Noise?

Perlin noise

 Modifications

Noise Characteristics

- Random
  - No correlation between distant values
- Repeatable/deterministic
  - Same argument always produces same value
- Band-limited
  - Most energy in one octave (e.g. between f & 2f)
What is this Noise?

Perlin noise

Noise Characteristics

• Random
  • No correlation between distant values

• Repeatable/deterministic
  • Same argument always produces same value

• Band-limited
  • Most energy in one octave (e.g. between f & 2f)
Noise Characteristics

- Random
  - No correlation between distant values
- Repeatable/deterministic
  - Same argument always produces same value
- Band-limited
  - Most energy in one octave (e.g. between f & 2f)
Gradient Noise

- Original Perlin noise [Perlin, 1985]
- Perlin Improved noise [Perlin, 2002]
  - Lattice based
    - Value=0 at integer lattice points
    - Gradient defined at integer lattice
    - Interpolate between
  - 1/2 to 1 cycle each unit
Gradient Noise

- Original Perlin noise [Perlin, 1985]
- Perlin Improved noise [Perlin, 2002]
- *Lattice* based
  - Value=0 at integer lattice points
  - Gradient defined at integer lattice
  - Interpolate between
- 1/2 to 1 cycle each unit
**Gradient Noise**

- Original Perlin noise [Perlin, 1985]
- Perlin Improved noise [Perlin, 2002]
- *Lattice* based
  - Value=0 at integer lattice points
  - Gradient defined at integer lattice
  - Interpolate between
- 1/2 to 1 cycle each unit
Value Noise

- Lattice based
  - Value defined at integer lattice points
  - Interpolate between
- At most 1/2 cycle each unit
  - Significant low-frequency content
- Easy hardware implementation with lower quality

![Linear Interp](chart1.png)  ![Cubic Interp](chart2.png)
What is this Noise?

Perlin noise

Value Noise

- Lattice based
  - Value defined at integer lattice points
  - Interpolate between
- At most 1/2 cycle each unit
  - Significant low-frequency content
- Easy hardware implementation with lower quality

Linear Interp

Cubic Interp
Value Noise

- Lattice based
  - Value defined at integer lattice points
  - Interpolate between
- At most 1/2 cycle each unit
  - Significant low-frequency content
- Easy hardware implementation with lower quality

![Graphs showing Linear Interp and Cubic Interp]
Value Noise

- Lattice based
  - Value defined at integer lattice points
  - Interpolate between
- At most 1/2 cycle each unit
  - Significant low-frequency content
- Easy hardware implementation with lower quality

![Linear Interpolated Noise](chart1)

![Cubic Interpolated Noise](chart2)

Linear Interp  Cubic Interp
Hardware Noise

• Value noise
  • PixelFlow [Lastra et al., 1995]
  • *Perlin Noise* Pixel Shaders [Hart, 2001]
  • Noise textures

• Gradient noise
  • Hardware [Perlin, 2001]
  • Complex composition [Perlin, 2004]
  • Shader implementation [Green, 2005]
Outline

What is this Noise?

Perlin noise

Modifications
Noise Details

• Subclass of *gradient noise*
  • Original Perlin
  • Perlin Improved
  • All of our proposed modifications
Find the Lattice

• Lattice-based noise: must find nearest lattice points
  • Point \( \vec{p} = (\vec{p}^x, \vec{p}^y, \vec{p}^z) \)
  • has integer lattice location \( \vec{p}_i = ([\vec{p}^x], [\vec{p}^y], [\vec{p}^z]) = (X, Y, Z) \)
  • and fractional location in cell \( \vec{p}_f = \vec{p} - \vec{p}_i = (x, y, z) \)
Find the Lattice

- Lattice-based noise: must find nearest lattice points
- Point $\vec{p} = (\vec{p}^x, \vec{p}^y, \vec{p}^z)$
  - has integer lattice location
    $\vec{p}_i = (\lfloor \vec{p}^x \rfloor, \lfloor \vec{p}^y \rfloor, \lfloor \vec{p}^z \rfloor) = (X, Y, Z)$
  - and fractional location in cell
    $\vec{p}_f = \vec{p} - \vec{p}_i = (x, y, z)$
Find the Lattice

- Lattice-based noise: must find nearest lattice points
- Point $\vec{p} = (\vec{p}^x, \vec{p}^y, \vec{p}^z)$
- has integer lattice location
  $$\vec{p}_i = (\lfloor\vec{p}^x\rfloor, \lfloor\vec{p}^y\rfloor, \lfloor\vec{p}^z\rfloor) = (X, Y, Z)$$
- and fractional location in cell
  $$\vec{p}_f = \vec{p} - \vec{p}_i = (x, y, z)$$
What is this Noise?

Perlin noise

Modifications

Find the Lattice

- Lattice-based noise: must find nearest lattice points
- Point $\vec{p} = (\vec{p}^x, \vec{p}^y, \vec{p}^z)$
- has integer lattice location $\vec{p}_i = (\lfloor \vec{p}^x \rfloor, \lfloor \vec{p}^y \rfloor, \lfloor \vec{p}^z \rfloor) = (X, Y, Z)$
- and fractional location in cell $\vec{p}_f = \vec{p} - \vec{p}_i = (x, y, z)$
What is this Noise?

Perlin noise

Modifications

---

**Gradient**

- Random vector at each lattice point is a function of $\vec{p}_i$

  $$g(\vec{p}_i)$$

- A function with that gradient

  $$\text{grad}(\vec{p}) = g(\vec{p}_i) \cdot \vec{p}_f$$

  $$= g^x(\vec{p}_i) \cdot x + g^y(\vec{p}_i) \cdot y + g^z(\vec{p}_i) \cdot z$$
Gradient

- Random vector at each lattice point is a function of $\vec{p}_i$

  $$g(\vec{p}_i)$$

- A function with that gradient

  $$\text{grad}(\vec{p}) = g(\vec{p}_i) \cdot \vec{p}_f$$

  $$= g^x(\vec{p}_i) \cdot x + g^y(\vec{p}_i) \cdot y + g^z(\vec{p}_i) \cdot z$$
Gradient

- Random vector at each lattice point is a function of $\vec{p}_i$

\[ g(\vec{p}_i) \]

- A function with that gradient

\[ \text{grad}(\vec{p}) = g(\vec{p}_i) \cdot \vec{p}_f \]
\[ = g^x(\vec{p}_i) \cdot x + g^y(\vec{p}_i) \cdot y + g^z(\vec{p}_i) \cdot z \]
Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\bar{p})$ is influenced by
  $\bar{p}_i + (0, 0)$, $\bar{p}_i + (0, 1)$, $\bar{p}_i + (1, 0)$, $\bar{p}_i + (1, 1)$
- Linear interpolation
  $\text{lerp}(t, a, b) = (1 - t) \cdot a + t \cdot b$
- Smooth interpolation
Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\vec{p})$ is influenced by
  $$\vec{p}_i + (0, 0); \vec{p}_i + (0, 1); \vec{p}_i + (1, 0); \vec{p}_i + (1, 1)$$
- Linear interpolation
  $$\text{lerp}(t, a, b) = (1 - t) \ a + t \ b$$
- Smooth interpolation
Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\vec{p})$ is influenced by
  $\vec{p}_i + (0,0)$; $\vec{p}_i + (0,1)$; $\vec{p}_i + (1,0)$; $\vec{p}_i + (1,1)$
- Linear interpolation
  - $\text{lerp}(t, a, b) = (1 - t) \cdot a + t \cdot b$
- Smooth interpolation
Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\vec{p})$ is influenced by
  $\vec{p}_i + (0, 0); \vec{p}_i + (0, 1); \vec{p}_i + (1, 0); \vec{p}_i + (1, 1)$
- Linear interpolation
  - $\text{lerp}(t, a, b) = (1 - t) \ a + t \ b$
- Smooth interpolation
Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\vec{p})$ is influenced by $\vec{p}_i + (0, 0); \vec{p}_i + (0, 1); \vec{p}_i + (1, 0); \vec{p}_i + (1, 1)$
- Linear interpolation
  - $\text{lerp}(t, a, b) = (1 - t) \ a + t \ b$
- Smooth interpolation
Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\vec{p})$ is influenced by
  $\vec{p}_i + (0, 0) ; \vec{p}_i + (0, 1) ; \vec{p}_i + (1, 0) ; \vec{p}_i + (1, 1)$
- Linear interpolation
  - $\text{lerp}(t, a, b) = (1 - t) \ a + t \ b$
- Smooth interpolation
  - $\text{fade}(t) = \begin{cases} 
  3t^2 - 2t^3 
  & \text{for original noise} \\
  10t^3 - 15t^4 + 6t^5 
  & \text{for improved noise}
\end{cases}$
  - $\text{flerp}(t) = \text{lerp}(\text{fade}(t), a, b)$
Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\vec{p})$ is influenced by
  $\vec{p}_i + (0, 0); \vec{p}_i + (0, 1); \vec{p}_i + (1, 0); \vec{p}_i + (1, 1)$
- Linear interpolation
  - $\text{lerp}(t, a, b) = (1 - t) \cdot a + t \cdot b$
- Smooth interpolation
  - $\text{fade}(t) = \begin{cases} 3t^2 - 2t^3 & \text{for original noise} \\ 10t^3 - 15t^4 + 6t^5 & \text{for improved noise} \end{cases}$
  - $\text{flerp}(t) = \text{lerp}(\text{fade}(t), a, b)$
What is this Noise?

Perlin noise

Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\vec{p})$ is influenced by $
  \vec{p}_i + (0, 0) ; \vec{p}_i + (0, 1) ; \vec{p}_i + (1, 0) ; \vec{p}_i + (1, 1)$
- Linear interpolation
  - $\text{lerp}(t, a, b) = (1 - t) a + t b$
- Smooth interpolation
  - $\text{fade}(t) = \begin{cases} 
  3t^2 - 2t^3 & \text{for original noise} \\
  10t^3 - 15t^4 + 6t^5 & \text{for improved noise} 
  \end{cases}$
  - $\text{flerp}(t) = \text{lerp}(\text{fade}(t), a, b)$
Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\vec{p})$ is influenced by
  $\vec{p}_i + (0, 0)$; $\vec{p}_i + (0, 1)$; $\vec{p}_i + (1, 0)$; $\vec{p}_i + (1, 1)$
- Linear interpolation
  - $\text{lerp}(t, a, b) = (1 - t) a + t b$
- Smooth interpolation
  - $\text{fade}(t) = \begin{cases} 
    3t^2 - 2t^3 & \text{for original noise} \\
    10t^3 - 15t^4 + 6t^5 & \text{for improved noise}
  \end{cases}$
  - $\text{flerp}(t) = \text{lerp}(\text{fade}(t), a, b)$
Interpolate

- Interpolate nearest $2^n$ gradient functions
- 2D $\text{noise}(\vec{p})$ is influenced by $\vec{p}_i + (0, 0)$; $\vec{p}_i + (0, 1)$; $\vec{p}_i + (1, 0)$; $\vec{p}_i + (1, 1)$
- Linear interpolation
  - $\text{lerp}(t, a, b) = (1 - t) a + t b$
- Smooth interpolation
  - $\text{fade}(t) = \begin{cases} 3t^2 - 2t^3 & \text{for original noise} \\ 10t^3 - 15t^4 + 6t^5 & \text{for improved noise} \end{cases}$
  - $\text{flerp}(t) = \text{lerp}(\text{fade}(t), a, b)$
Hash

- n-D gradient function built from 1D components

\[ g(\vec{p}_i) \]

- Both original and improved use a permutation table *hash*
- Original: \( g \) is a table of unit vectors
- Improved: \( g \) is derived from bits of final hash
Hash

- n-D gradient function built from 1D components

\[ g(\text{hash}(X, Y, Z)) \]

- Both original and improved use a permutation table \textit{hash}
- Original: \( g \) is a table of unit vectors
- Improved: \( g \) is derived from bits of final hash
Hash

- n-D gradient function built from 1D components

\[ g(\text{hash}(Z + \text{hash}(X, Y))) \]

- Both original and improved use a permutation table \text{hash}
- Original: \( g \) is a table of unit vectors
- Improved: \( g \) is derived from bits of final hash
Hash

• n-D gradient function built from 1D components

\[
g(\text{hash}(Z + \text{hash}(Y + \text{hash}(X))))
\]

• Both original and improved use a permutation table \( \text{hash} \)
• Original: \( g \) is a table of unit vectors
• Improved: \( g \) is derived from bits of final hash
Hash

- n-D gradient function built from 1D components
  \[ g(\text{hash}(Z + \text{hash}(Y + \text{hash}(X)))) \]
- Both original and improved use a permutation table \textit{hash}
  - Original: \( g \) is a table of unit vectors
  - Improved: \( g \) is derived from bits of final hash
Hash

- n-D gradient function built from 1D components

\[ g(\text{hash}(Z + \text{hash}(Y + \text{hash}(X)))) \]

- Both original and improved use a permutation table \text{hash}
- Original: \( g \) is a table of unit vectors
- Improved: \( g \) is derived from bits of final hash
Outline

What is this Noise?

Perlin noise

Modifications
    Corner Gradients
    Factorization
    Hash
Gradient Vectors of n-D Noise

- Original: on the surface of a n-sphere
  - Found by hash of $\vec{p}_i$ into gradient table
- Improved: at the edges of an n-cube
  - Found by decoding bits of hash of $\vec{p}_i$
Gradient Vectors of n-D Noise

- Original: on the surface of a n-sphere
  - Found by hash of $\vec{p}_i$ into gradient table
- Improved: at the edges of an n-cube
  - Found by decoding bits of hash of $\vec{p}_i$
Gradients of noise(x,y,0) or noise(x,0)

- Why?
  - Cheaper low-D noise matches slice of higher-D
  - Reuse textures (for full noise or partial computation)
- Original: new short gradient vectors
- Improved: gradients in new directions
  - Possibly including 0 gradient vector!
Gradients of noise\((x,y,0)\) or \(\text{noise}(x,0)\)

- Why?
  - Cheaper low-D noise matches slice of higher-D
  - Reuse textures (for full noise or partial computation)
- Original: new short gradient vectors
- Improved: gradients in new directions
  - Possibly including 0 gradient vector!
Gradients of $\text{noise}(x,y,0)$ or $\text{noise}(x,0)$

- **Why?**
  - Cheaper low-D noise matches slice of higher-D
  - Reuse textures (for full noise or partial computation)
- **Original:** new short gradient vectors
- **Improved:** gradients in new directions
  - Possibly including 0 gradient vector!
Solution?

- Observe: use gradient function, not vector alone

\[ \text{grad} = g^x \cdot x + g^y \cdot y + g^z \cdot z \]

- In any integer plane, fractional \( z = 0 \)

\[ \text{grad} = g^x \cdot x + g^y \cdot y + 0 \]

- Any choice keeping projection of vectors the same will work
  - Improved noise uses cube edge centers
Solution?

- Observe: use gradient function, not vector alone

\[ \text{grad} = g^x \ x + g^y \ y + g^z \ z \]

- In any integer plane, fractional \( z = 0 \)

\[ \text{grad} = g^x \ x + g^y \ y + 0 \]

- Any choice keeping projection of vectors the same will work
  - Improved noise uses cube edge centers
  - Instead use cube corners!
Solution?

- Observe: use gradient function, not vector alone

\[ \text{grad} = g^x \cdot x + g^y \cdot y + g^z \cdot z \]

- In any integer plane, fractional \( z = 0 \)

\[ \text{grad} = g^x \cdot x + g^y \cdot y + 0 \]

- Any choice keeping projection of vectors the same will work
  - Improved noise uses cube edge centers
  - Instead use cube corners!
Solution?

• Observe: use gradient function, not vector alone

\[ \text{grad} = g^x x + g^y y + g^z z \]

• In any integer plane, fractional \( z = 0 \)

\[ \text{grad} = g^x x + g^y y + 0 \]

• Any choice keeping projection of vectors the same will work
  • Improved noise uses cube edge centers
  • Instead use cube corners!
Solution?

- Observe: use gradient function, not vector alone

\[ \text{grad} = g_x x + g_y y + g_z z \]

- In any integer plane, fractional \( z = 0 \)

\[ \text{grad} = g_x x + g_y y + 0 \]

- Any choice keeping projection of vectors the same will work
  - Improved noise uses cube edge centers
  - Instead use cube corners!
Corner Gradients

- Simple binary selection from hash bits
  ±x, ±y, ±z
- Perlin mentions “clumping” for corner gradient selection
  - Not very noticeable in practice
  - Already happens in any integer plane of improved noise
Corner Gradients

• Simple binary selection from hash bits
  \[ \pm x, \pm y, \pm z \]

• Perlin mentions “clumping” for corner gradient selection
  • Not very noticeable in practice
  • Already happens in any integer plane of improved noise

Edge Centers  Corner
Separable Computation

- Like to store computation in texture
  - Texture sampling 3-4x highest frequency

- 1D & 2D OK size, 3D gets big, 4D impossible
- Factor into lower-D textures
Separable Computation

- Like to store computation in texture
  - Texture sampling 3-4x highest frequency

- 1D & 2D OK size, 3D gets **big**, 4D impossible
- Factor into lower-D textures
  - (e.g. write noise(\vec{p}_x, \vec{p}_y, \vec{p}_z) as several 2D terms)
  \[
  \text{noise}(\vec{p}_x, \vec{p}_y, \vec{p}_z) = \text{flerp}(z, +z + (z - 1))
  \]
Separable Computation

- Like to store computation in texture
  - Texture sampling 3-4× highest frequency
- 1D & 2D OK size, 3D gets **big**, 4D impossible
- Factor into lower-D textures
  - (e.g. write \( \text{noise}(\vec{p}^x, \vec{p}^y, \vec{p}^z) \) as several x/y terms)
    \[
    \text{noise}(\vec{p}^x, \vec{p}^y, \vec{p}^z) = \text{flerp}(z, + * z
    + * (z - 1))
    \]
Separable Computation

- Like to store computation in texture
  - Texture sampling 3-4x highest frequency
- 1D & 2D OK size, 3D gets **big**, 4D impossible
- Factor into lower-D textures
  - (e.g. write \( \text{noise}(\vec{p}^x, \vec{p}^y, \vec{p}^z) \) as several x/y terms)

\[
\text{noise}(\vec{p}^x, \vec{p}^y, \vec{p}^z) = \text{flerp}(z, + * z + * (z - 1))
\]
Separable Computation

- Like to store computation in texture
  - Texture sampling 3-4x highest frequency

- 1D & 2D OK size, 3D gets big, 4D impossible
- Factor into lower-D textures
  - (e.g. write \( \text{noise}(\vec{p}^x, \vec{p}^y, \vec{p}^z) \) as several x/y terms)
  \[
  \text{noise}(\vec{p}^x, \vec{p}^y, \vec{p}^z) = \text{flerp}(z, \text{xyz-term} + \text{xyz-term} \times z \\
  \text{xyz-term} + \text{xyz-term} \times (z - 1))
  \]
Separable Computation

- Like to store computation in texture
  - Texture sampling 3-4x highest frequency
- 1D & 2D OK size, 3D gets big, 4D impossible
- Factor into lower-D textures
  - (e.g. write $\text{noise}(\vec{p}^x, \vec{p}^y, \vec{p}^z)$ as several x/y terms)

$$
\text{noise}(\vec{p}^x, \vec{p}^y, \vec{p}^z) = \text{flerp}(z, \text{xy-term}(Z_0) + \text{xy-term}(Z_0) \times z \\
\text{xy-term}(Z_1) + \text{xy-term}(Z_1) \times (z - 1))
$$
What is this Noise? Perlin noise

Factorization Details

\[
\text{noise}(\vec{p}) = \text{flerp}(z, z\text{const}(\vec{p}^x, \vec{p}^y, Z_0) + z\text{grad}(\vec{p}^x, \vec{p}^y, Z_0) \ast z, \\
z\text{const}(\vec{p}^x, \vec{p}^y, Z_1) + z\text{grad}(\vec{p}^x, \vec{p}^y, Z_1) \ast (z - 1))
\]

- With nested hash,
  \[
  z\text{const}(\vec{p}^x, \vec{p}^y, Z_0) = z\text{const}(\vec{p}^x, \vec{p}^y + \text{hash}(Z_0)) \\
z\text{grad}(\vec{p}^x, \vec{p}^y, Z_0) = z\text{grad}(\vec{p}^x, \vec{p}^y + \text{hash}(Z_0))
  \]

- With corner gradients, \(z\text{const} = \text{noise}\)!
Factorization Details

\[
\text{noise}(\vec{p}) = \text{flerp}(z, z\text{const}(\vec{p}_x, \vec{p}_y, Z_0) + z\text{grad}(\vec{p}_x, \vec{p}_y, Z_0) \times z, \\
z\text{const}(\vec{p}_x, \vec{p}_y, Z_1) + z\text{grad}(\vec{p}_x, \vec{p}_y, Z_1) \times (z - 1))
\]

- With nested hash,

\[
\begin{align*}
    z\text{const}(\vec{p}_x, \vec{p}_y, Z_0) &= z\text{const}(\vec{p}_x, \vec{p}_y + \text{hash}(Z_0)) \\
    z\text{grad} (\vec{p}_x, \vec{p}_y, Z_0) &= z\text{grad} (\vec{p}_x, \vec{p}_y + \text{hash}(Z_0))
\end{align*}
\]

- With corner gradients, \(z\text{const} = \text{noise}\)!
What is this Noise? Perlin noise

Factorization Details

\[
\text{noise}(\vec{p}) = \text{flerp}(z, \text{zconst}(\vec{p}^x, \vec{p}^y, Z_0) + \text{zgrad}(\vec{p}^x, \vec{p}^y, Z_0) \ast z, \\
\text{zconst}(\vec{p}^x, \vec{p}^y, Z_1) + \text{zgrad}(\vec{p}^x, \vec{p}^y, Z_1) \ast (z - 1))
\]

- With nested hash,

\[
\text{zconst}(\vec{p}^x, \vec{p}^y, Z_0) = \text{zconst}(\vec{p}^x, \vec{p}^y + \text{hash}(Z_0)) \\
\text{zgrad}(\vec{p}^x, \vec{p}^y, Z_0) = \text{zgrad}(\vec{p}^x, \vec{p}^y + \text{hash}(Z_0))
\]

- With corner gradients, \(z\text{const} = \text{noise}\)!
Perlin’s Hash

- 256-element *permutation array*
  - Turns each integer 0-255 into a different integer 0-255
- Chained lookups
  \[ g(hash(Z + hash(Y + hash(X))))) \]
- Must compute for each lattice point around \( \vec{p} \)
- Even with a 2D \( hash(Y + hash(X)) \) texture, that’s...
What is this Noise? Perlin noise

### Perlin’s Hash

- 256-element *permutation array*
  - Turns each integer 0-255 into a different integer 0-255
- Chained lookups
  \[ g(\text{hash}(Z + \text{hash}(Y + \text{hash}(X)))) \]
- Must compute for each lattice point around \( \vec{p} \)
- Even with a 2D \( \text{hash}(Y + \text{hash}(X)) \) texture, that’s
  \# 3 hash lookups for 1D noise
Perlin’s Hash

- 256-element *permutation array*
  - Turns each integer 0-255 into a different integer 0-255
- Chained lookups
  \[ g(\text{hash}(Z + \text{hash}(Y + \text{hash}(X)))) \]
- Must compute for each lattice point around \( \vec{p} \)
- Even with a 2D \( \text{hash}(Y + \text{hash}(X)) \) texture, that’s
  - 2 hash lookups for 1D noise
  - 4 hash lookups for 2D noise
  - 12 hash lookups for 3D noise
  - 20 hash lookups for 4D noise
Perlin’s Hash

- 256-element *permutation array*
  - Turns each integer 0-255 into a different integer 0-255
- Chained lookups
  \[ g(hash(Z + hash(Y + hash(X))))) \]
- Must compute for each lattice point around \( \vec{p} \)
- Even with a 2D \( hash(Y + hash(X)) \) texture, that’s
  - 2 hash lookups for 1D noise
  - 4 hash lookups for 2D noise
  - 12 hash lookups for 3D noise
  - 20 hash lookups for 4D noise
Perlin’s Hash

- 256-element *permutation array*
  - Turns each integer 0-255 into a different integer 0-255
- Chained lookups
  \[ g(\text{hash}(Z + \text{hash}(Y + \text{hash}(X)))) \]
- Must compute for each lattice point around \( \vec{p} \)
- Even with a 2D \( \text{hash}(Y + \text{hash}(X)) \) texture, that’s
  - 2 hash lookups for 1D noise
  - 4 hash lookups for 2D noise
  - 12 hash lookups for 3D noise
  - 20 hash lookups for 4D noise
Perlin’s Hash

- 256-element *permutation array*
  - Turns each integer 0-255 into a different integer 0-255
- Chained lookups
  \[ g(hash(Z + hash(Y + hash(X)))) \]
- Must compute for each lattice point around \( \vec{p} \)
- Even with a 2D \( hash(Y + hash(X)) \) texture, that’s
  - 2 hash lookups for 1D noise
  - 4 hash lookups for 2D noise
  - 12 hash lookups for 3D noise
  - 20 hash lookups for 4D noise
Perlin’s Hash

- 256-element *permutation array*
  - Turns each integer 0-255 into a different integer 0-255
- Chained lookups
  \[ g(\text{hash}(Z + \text{hash}(Y + \text{hash}(X)))) \]
- Must compute for each lattice point around \( \vec{p} \)
- Even with a 2D \( \text{hash}(Y + \text{hash}(X)) \) texture, that’s
  - 2 hash lookups for 1D noise
  - 4 hash lookups for 2D noise
  - 12 hash lookups for 3D noise
  - 20 hash lookups for 4D noise
Perlin’s Hash

- 256-element *permutation array*
  - Turns each integer 0-255 into a different integer 0-255
- Chained lookups
  \[ g(hash(Z + hash(Y + hash(X)))) \]
- Must compute for each lattice point around \( \vec{p} \)
- Even with a 2D \( hash(Y + hash(X)) \) texture, that’s
  - 2 hash lookups for 1D noise
  - 4 hash lookups for 2D noise
  - 12 hash lookups for 3D noise
  - 20 hash lookups for 4D noise
Alternative Hash

- Many choices; I kept 1D chaining
- Desired features
  - Low correlation of hash output for nearby inputs
  - Computable without lookup
- Use a random number generator?
  - Seed
  - Successive calls give uncorrelated values
What is this Noise? Perlin noise

Alternate Hash

- Many choices; I kept 1D chaining
- Desired features
  - Low correlation of hash output for nearby inputs
  - Computable without lookup
- Use a random number generator?
  - Seed
  - Successive calls give uncorrelated values
Alternative Hash

• Many choices; I kept 1D chaining
• Desired features
  • Low correlation of hash output for nearby inputs
  • Computable without lookup
• Use a random number generator?
  • Seed
  • Successive calls give uncorrelated values
Random Number Generator Hash

- Hash argument is seed
  - Most RNG are highly correlated for nearby seeds
- Hash argument is number of times to call
  - Most RNG are expensive (or require n calls) to get $n^{th}$ number
  - Should noise(30) be 30 times slower than noise(1)?

 permute table

 hash using seed=X
Random Number Generator Hash

- Hash argument is seed
  - Most RNG are highly correlated for nearby seeds
- Hash argument is number of times to call
  - Most RNG are expensive (or require n calls) to get $n^{th}$ number
  - Should $\text{noise}(30)$ be 30 times slower than $\text{noise}(1)$?

permute table

hash using $X^{th}$ random number
What is this Noise? Perlin noise

Blum-Blum Shub

\[ x_{n+1} = x_i^2 \mod M \]

\[ M = \text{product of two large primes} \]

- Uncorrelated for nearby seeds...
- But large M is bad for hardware...
- But reasonable results for smaller M...
- And square and mod is simple to compute!

523*527
What is this Noise? Perlin noise

Blum-Blum Shub

\[ x_{n+1} = x_i^2 \mod M \]

\[ M = \text{product of two large primes} \]

- Uncorrelated for nearby seeds...
- But large \( M \) is bad for hardware...
- But reasonable results for smaller \( M \)...
- And square and mod is simple to compute!

523*527
Blum-Blum Shub

\[ x_{n+1} = x_i^2 \mod M \]

\[ M = \text{product of two large primes} \]

- Uncorrelated for nearby seeds...
- But large \( M \) is bad for hardware...
- But reasonable results for smaller \( M \)...
- And square and mod is simple to compute!

523*527
Blum-Blum Shub

\[ x_{n+1} = x_i^2 \mod M \]

\[ M = \text{product of two large primes} \]

- Uncorrelated for nearby seeds...
- But large \( M \) is bad for hardware...
- But reasonable results for smaller \( M \)...
- And square and mod is simple to compute!
Blum-Blum Shub

\[ x_{n+1} = x_i^2 \mod M \]

\[ M = \text{product of two large primes} \]

- Uncorrelated for nearby seeds...
- But large M is bad for hardware...
- But reasonable results for smaller M...
- And square and mod is simple to compute!
Modified Noise

- Square and mod hash
  - \( M = 61 \)
- Corner gradient selection
  - One 2D texture for both 1D and 2D
- Factor
  - Construct 3D and 4D from 2 or 4 2D texture lookups
What is this Noise? Perlin noise

Comparison

Perlin original

Perlin improved

Corner gradients

Corner + Hash
Using Noise

- 3D noise
- 3D turbulence
- Wood
- Marble
What is this Noise? Perlin noise

Brook for GPUs: Stream computing on graphics hardware.
ACM Transactions on Graphics, 23(3).

Implementing improved Perlin noise.
Addison-Wesley.

Geometry images.

Perlin noise pixel shaders.
SIGGRAPH/EUROGRAPHICS, ACM, New York.


Noise hardware.
In Olano, M., editor, *Real-Time Shading SIGGRAPH Course Notes*.

Improving noise.

Implementing improved Perlin noise.
In Fernando, R., editor, *GPU Gems*, chapter 5. Addison-Wesley.

A real-time procedural shading system for programmable graphics hardware.
In *Proc. ACM SIGGRAPH*. 
Real-time procedural textures.