Combining Computer Vision and Physics Simulations using GPGPU

SIGGRAPH2007

Justin Hensley

John Isidoro

Arcot J. Preetham

Graphics Product Group

AMD

Overview

AMD

Overview

- Motivation

Overview

- Motivation
- GPGP-what?

Overview

- Motivation
- GPGP-what?
- Depth-extraction

Overview

AMD

- Motivation
- GPGP-what?
- Depth-extraction
- Rigid-body physics

Overview

- Motivation
- GPGP-what?
- Depth-extraction
- Rigid-body physics
- Putting it all together

Overview

- Motivation
- GPGP-what?
- Depth-extraction
- Rigid-body physics
- Putting it all together
- Demo

Overview

- Motivation
- GPGP-what?
- Depth-extraction
- Rigid-body physics
- Putting it all together
- Demo
- Conclusion and questions

Motivation

Motivation

- Enable novel input paradigms

Motivation

- Enable novel input paradigms
- Cameras are fairly ubiquitous
  - Would like to use the cameras for HCI

Motivation

- Enable novel input paradigms
- Cameras are fairly ubiquitous
  - Would like to use the cameras for HCI
- Computer vision requires large amounts of compute
  - Leverage computational power of Graphics Processors (GPUs)

Motivation

- Enable novel input paradigms
- Cameras are fairly ubiquitous
  - Would like to use the cameras for HCI
- Computer vision requires large amounts of compute
  - Leverage computational power of Graphics Processors (GPUs)
- Our Demo application
  - Use a camera to generate models that can interact with rigid-body physics simulation

GPGP-what???

AMD

GPGP-what???

AMD

- **G**eneral-**P**urpose Computation on **GPU**s

GPGP-what???

- **G**eneral-**P**urpose Computation on **GPU**s
- Modern GPUs have enormous computational power and memory bandwidth
  - AMD HD 2900 XT:
  - – 320 Scalar ALUs
  - – ~470 GFlops & 108 GB / sec of memory bandwidth

face tomorrow
SIGGRAPH2007

GPGP-what???

AMD

- **G**eneral-**P**urpose Computation on **GPU**s
- Modern GPUs have enormous computational power and memory bandwidth
  - AMD HD 2900 XT:
  - – 320 Scalar ALUs
  - – ~470 GFlops & 108 GB / sec of memory bandwidth
- Works best with data parallel algorithms
  - – Rendering, image processing, physical simulation etc.

GPGP-what???

AMD

- **G**eneral-**P**urpose Computation on **GPU**s
- Modern GPUs have enormous computational power and memory bandwidth
  - AMD HD 2900 XT:
  - – 320 Scalar ALUs
  - – ~470 GFlops & 108 GB / sec of memory bandwidth
- Works best with data parallel algorithms
  - – Rendering, image processing, physical simulation etc.
- Used AMD's Close to the Metal (CTM) SDK
  - – Easier to use than graphics APIs for GPGPU

Depth Extraction Overview

- Compute depth from pair of images
  - Variant of [wang06]
- Rectify images (requires calibrated cameras)
  - Scanlines in the one image correspond to scanlines in the other image
- Compute Cost Volume
- Aggregate Cost Volume
- Find best depth estimate

[wang06] Wang, L., Liao, M., Gong, M., Yang, R., and Nister, D. 2006. High-Quality Real-Time Stereo Using Adaptive Cost Aggregation and Dynamic Programming. 3DPVT'06

Image Rectification

AMD

Image Rectification

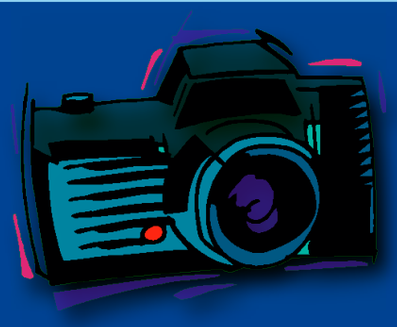
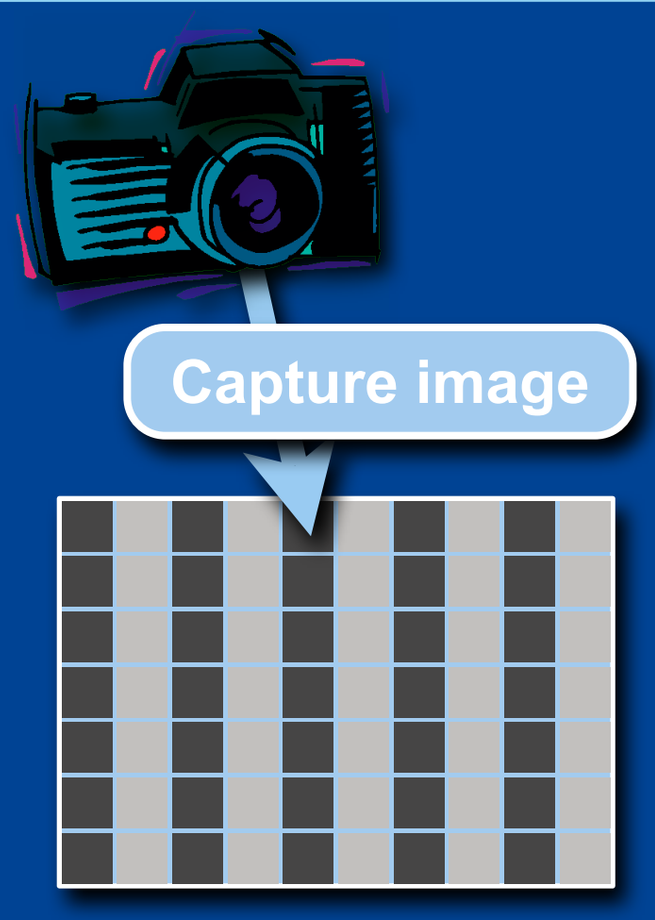


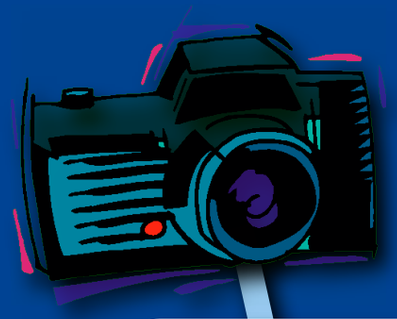
Image Rectification

AMD

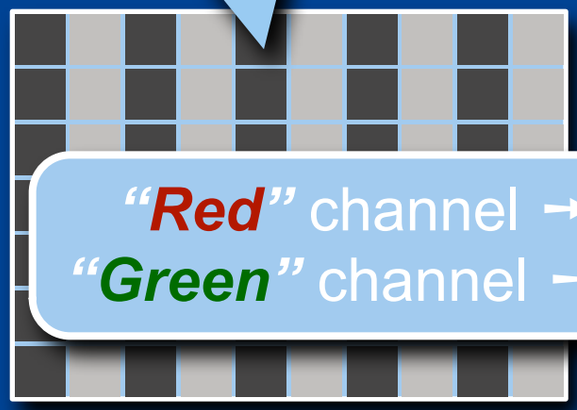


face tomorrow

Image Rectification

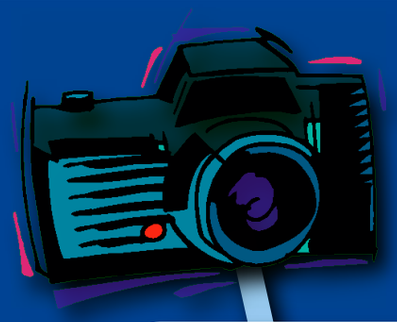


Capture image

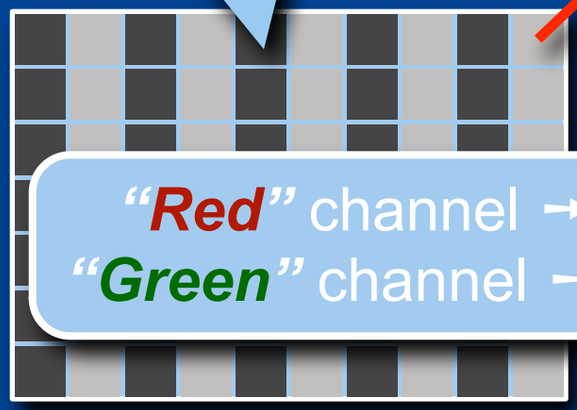


"Red" channel → **left** image
"Green" channel → **right** image

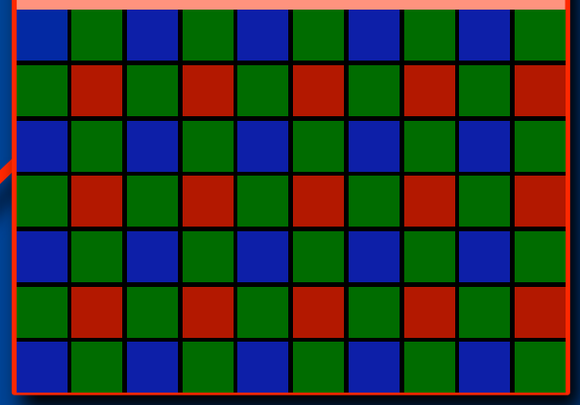
Image Rectification



Capture image



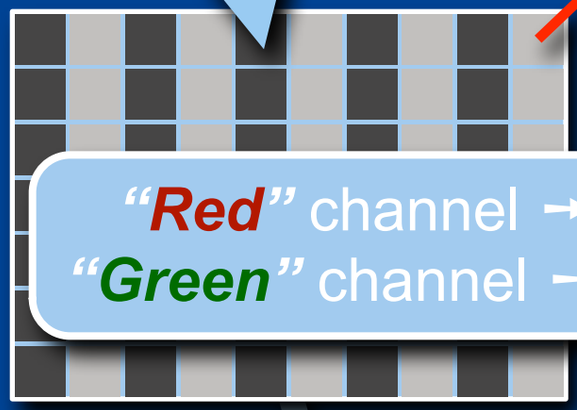
Bayer Pattern:



"Red" channel → **left** image
"Green" channel → **right** image

Image Rectification

Capture image



"*Red*" channel → **left** image
"*Green*" channel → **right** image

Separate & de-Bayer

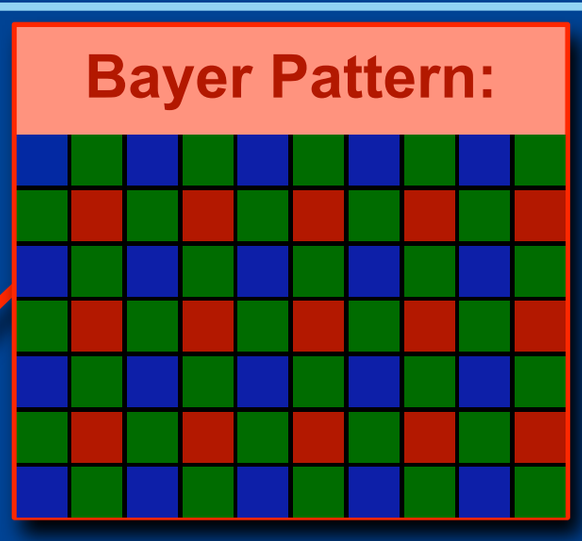


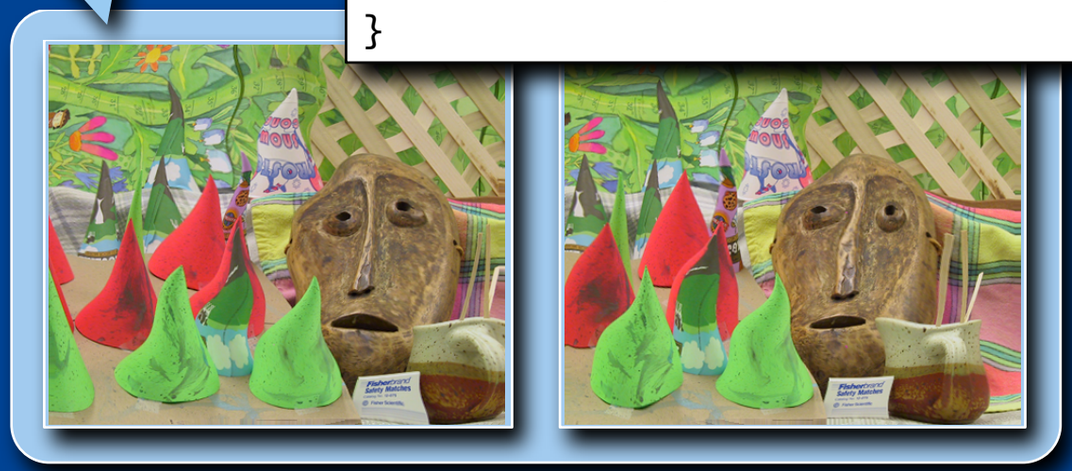
Image Rectification

Capture image

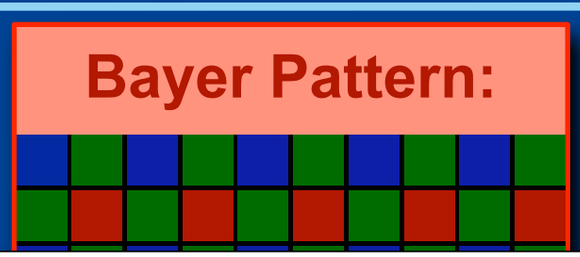
"Red" channel →

"Green" channel →

Separate & de-B



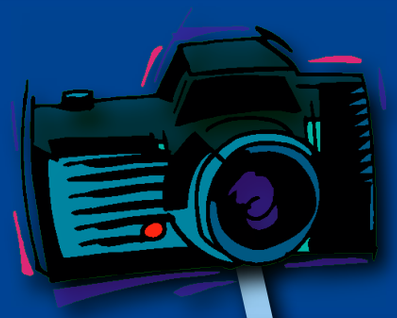
Bayer Pattern:



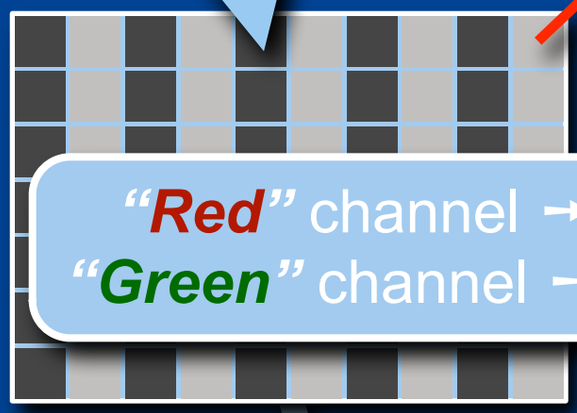
```
// Shader to split packed stereo image from BB into two images
TwoOutput_PS UnpackAndConvertStereoBayerImage( float2 vpos: VPOS )
{
   // compute location and offset
   float2 offset = 1.0 / gImageSize;
   float2 pos    = (vpos + 0.5) / gImageSize;

   // load BGGR Bayer pattern data
   float2 texB  = tex2D( gPackedImage, pos ).rg;
   float2 texG0 = tex2D( gPackedImage, pos + float2( offset.x, 0.0 ) ).rg;
   float2 texG1 = tex2D( gPackedImage, pos + float2( 0.0, offset.y ) ).rg;
   float2 texR  = tex2D( gPackedImage, pos + offset ).rg;

   // Interpolate Bayer data data and set outputs
   TwoOutput_PS outC;
   outC.Col0 = float4( texR[0], 0.5 * ( texG0[0] + texG1[0] ), texB[0], 1.0 );
   outC.Col1 = float4( texR[1], 0.5 * ( texG0[1] + texG1[1] ), texB[1], 1.0 );

   return outC;
}
```


face tomorrow
SIGGRAPH2007

Image Rectification



Capture image



"Red" channel →

"Green" channel →

Separate & de-B

Bayer Pattern:

```
// Shader to split packed stereo image from BB into two images
TwoOutput_PS UnpackAndConvertStereoBayerImage( float2 vpos: VPOS )
{
   // compute location and offset
   float2 offset = 1.0 / gImageSize;
   float2 pos    = (vpos + 0.5) / gImageSize;

   // load BGGR Bayer pattern data
   float2 texB  = tex2D( gPackedImage, pos ).rg;
   float2 texG0 = tex2D( gPackedImage, pos + float2( offset.x, 0.0 ) ).rg;
   float2 texG1 = tex2D( gPackedImage, pos + float2( 0.0, offset.y ) ).rg;
   float2 texR  = tex2D( gPackedImage, pos + offset ).rg;

   // Interpolate Bayer data data and set outputs
   TwoOutput_PS outC;
   outC.Col0 = float4( texR[0], 0.5 * ( texG0[0] + texG1[0] ), texB[0], 1.0 );
   outC.Col1 = float4( texR[1], 0.5 * ( texG0[1] + texG1[1] ), texB[1], 1.0 );

   return outC;
}
```

Image Rectification

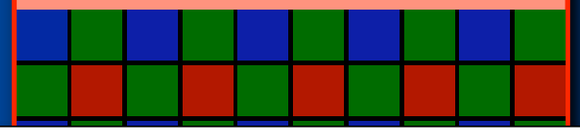
Capture image

"Red" channel →

"Green" channel →

Separate & de-B

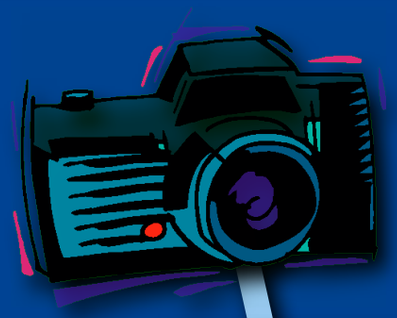
Bayer Pattern:



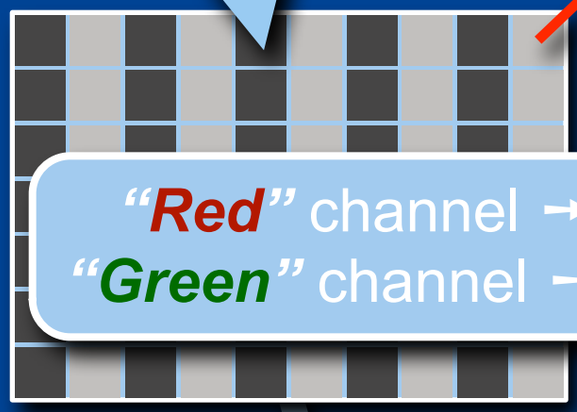
```
// Shader to split packed stereo image from BB into two images
TwoOutput_PS UnpackAndConvertStereoBayerImage( float2 vpos: VPOS )
{
   // compute location and offset
   float2 offset = 1.0 / gImageSize;
   float2 pos    = (vpos + 0.5) / gImageSize;

   // load BGGR Bayer pattern data
   float2 texB  = tex2D( gPackedImage, pos ).rg;
   float2 texG0 = tex2D( gPackedImage, pos + float2( offset.x, 0.0 ) ).rg;
   float2 texG1 = tex2D( gPackedImage, pos + float2( 0.0, offset.y ) ).rg;
   float2 texR  = tex2D( gPackedImage, pos + offset ).rg;

   // Interpolate Bayer data data and set outputs
   TwoOutput_PS outC;
   outC.Col0 = float4( texR[0], 0.5 * ( texG0[0] + texG1[0] ), texB[0], 1.0 );
   outC.Col1 = float4( texR[1], 0.5 * ( texG0[1] + texG1[1] ), texB[1], 1.0 );

   return outC;
}
```


face tomorrow
SIGGRAPH2007

Image Rectification



Capture image

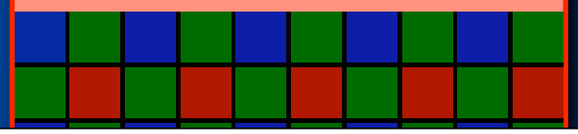


"*Red*" channel →

"*Green*" channel →

Separate & de-B

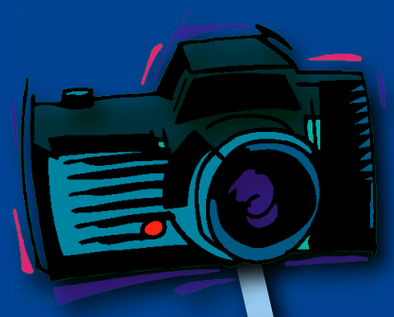
Bayer Pattern:



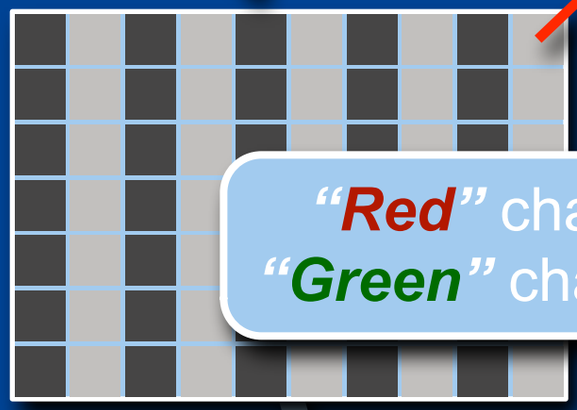
```
// Shader to split packed stereo image from BB into two images
TwoOutput_PS UnpackAndConvertStereoBayerImage( float2 vpos: VPOS )
{
   // compute location and offset
   float2 offset = 1.0 / gImageSize;
   float2 pos    = (vpos + 0.5) / gImageSize;

   // load BGGR Bayer pattern data
   float2 texB  = tex2D( gPackedImage, pos ).rg;
   float2 texG0 = tex2D( gPackedImage, pos + float2( offset.x, 0.0 ) ).rg;
   float2 texG1 = tex2D( gPackedImage, pos + float2( 0.0, offset.y ) ).rg;
   float2 texR  = tex2D( gPackedImage, pos + offset ).rg;

   // Interpolate Bayer data data and set outputs
   TwoOutput_PS outC;
   outC.Col0 = float4( texR[0], 0.5 * ( texG0[0] + texG1[0] ), texB[0], 1.0 );
   outC.Col1 = float4( texR[1], 0.5 * ( texG0[1] + texG1[1] ), texB[1], 1.0 );

   return outC;
}
```


face tomorrow
SIGGRAPH2007

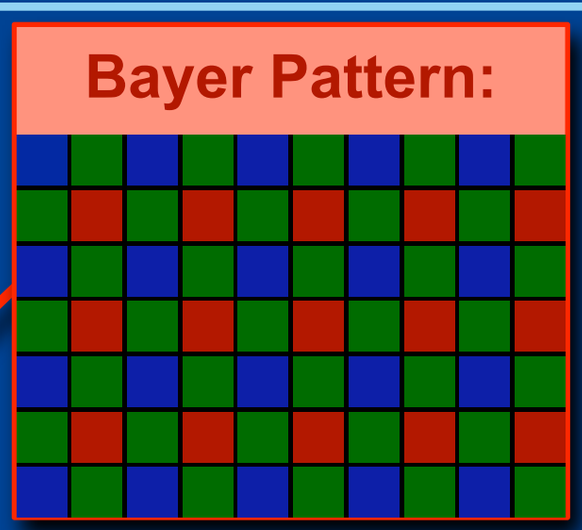
Image Rectification



Capture image



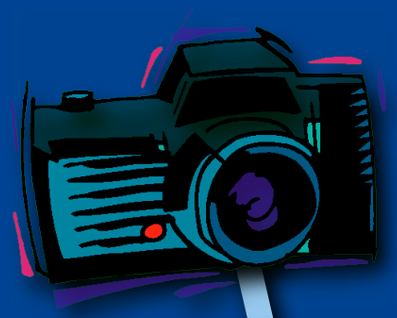
Bayer Pattern:



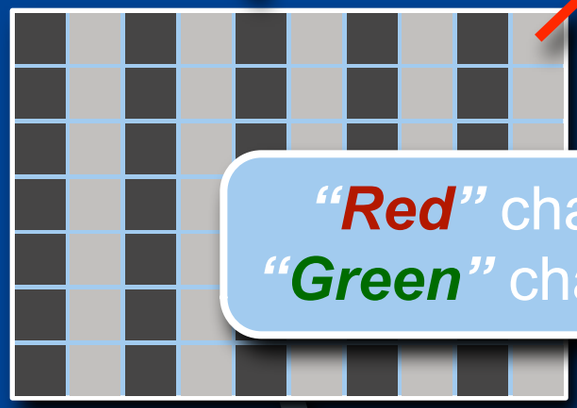
"*Red*" channel → **left** image
"*Green*" channel → **right** image

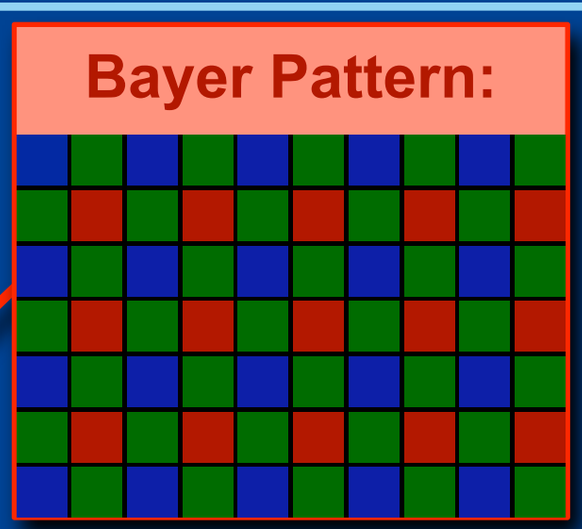
Separate & de-Bayer

Image Rectification



Capture image





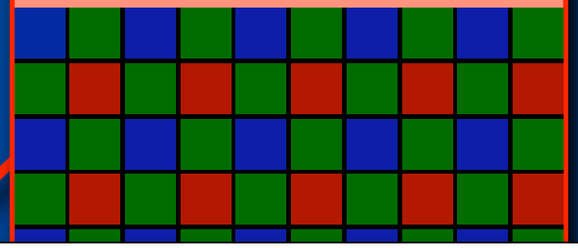
"*Red*" channel → **left** image
"*Green*" channel → **right** image

Separate & de-Bayer

Rectify

Image Rectification

Bayer Pattern:

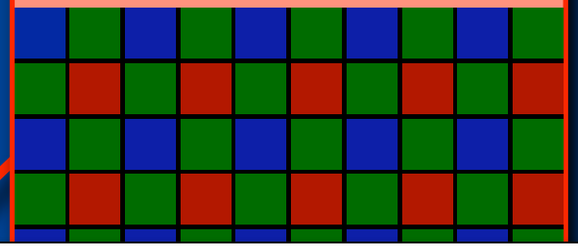


```
// Shader to perform warp on an image
// (bilinear warp done via filtering)
OneOutput_PS WarpImage( float2 vpos: VPOS )
{
   // compute scaled texcoord
   float2 pos = (vpos + 0.5) / gImageSize;

   float2 wtexel = tex2D( gWarpMap, pos ).rg;
   float2 wpos  = (wtexel + 0.5) / gMapSize;

   // bilinearly interpolate pixel at warped position
   OneOutput_PS outC;
   outC.Col0 = tex2D( gInput, wpos );;
   return outC;
}
```

Rectify

Image Rectification

Bayer Pattern:

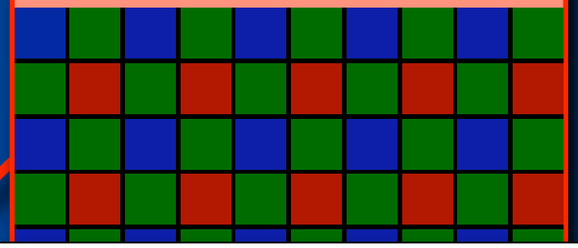


```
// Shader to perform warp on an image
// (bilinear warp done via filtering)
OneOutput_PS WarpImage( float2 vpos: VPOS )
{
   // compute scaled texcoord
   float2 pos = (vpos + 0.5) / gImageSize;

   float2 wtexel = tex2D( gWarpMap, pos ).rg;
   float2 wpos  = (wtexel + 0.5) / gMapSize;

   // bilinearly interpolate pixel at warped position
   OneOutput_PS outC;
   outC.Col0 = tex2D( gInput, wpos );;
   return outC;
}
```

Rectify

Image Rectification

Bayer Pattern:

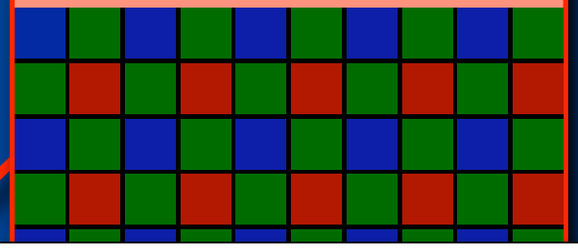


```
// Shader to perform warp on an image
// (bilinear warp done via filtering)
OneOutput_PS WarpImage( float2 vpos: VPOS )
{
   // compute scaled texcoord
   float2 pos = (vpos + 0.5) / gImageSize;

   float2 wtexel = tex2D( gWarpMap, pos ).rg;
   float2 wpos  = (wtexel + 0.5) / gMapSize;

   // bilinearly interpolate pixel at warped position
   OneOutput_PS outC;
   outC.Col0 = tex2D( gInput, wpos );;
   return outC;
}
```

Rectify

Image Rectification

Bayer Pattern:



```
// Shader to perform warp on an image
// (bilinear warp done via filtering)
OneOutput_PS WarpImage( float2 vpos: VPOS )
{
   // compute scaled texcoord
   float2 pos = (vpos + 0.5) / gImageSize;

   float2 wtexel = tex2D( gWarpMap, pos ).rg;
   float2 wpos  = (wtexel + 0.5) / gMapSize;

   // bilinearly interpolate pixel at warp position and output
   OneOutput_PS outC;
   outC.Col0 = tex2D( gInput, wpos );
   return outC;
}
```

Rectify

Computing the Cost Volume

AMD

Computing the Cost Volume

- Compute volume of candidate *disparity* costs

Computing the Cost Volume

- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images

Computing the Cost Volume

AMD

- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position

Computing the Cost Volume

AMD

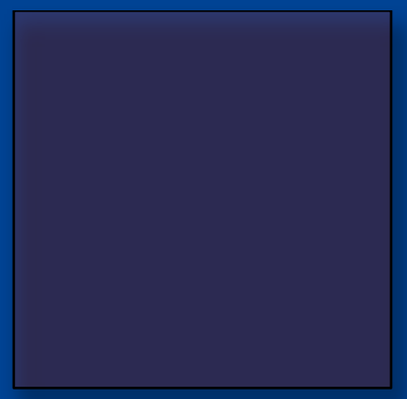
- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position

Computing the Cost Volume

- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position

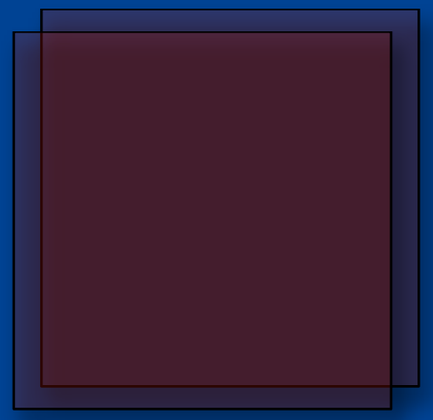
Computing the Cost Volume

- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position



Computing the Cost Volume

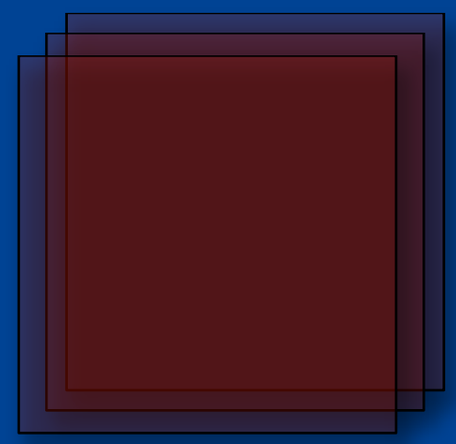
- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position



Computing the Cost Volume

AMD

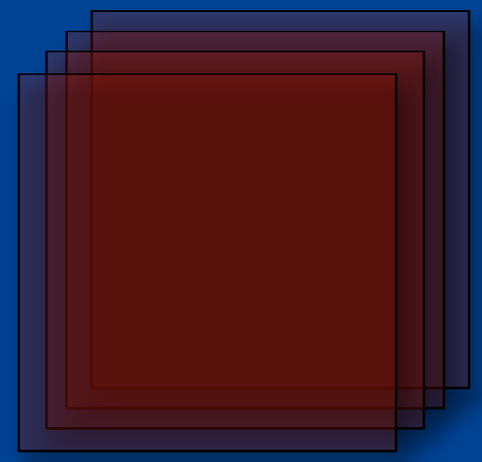
- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position



Computing the Cost Volume

AMD

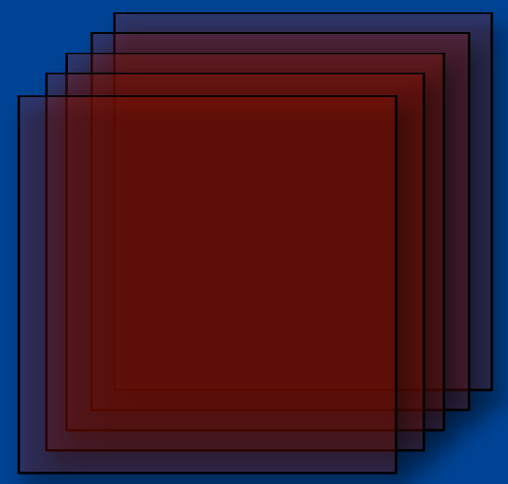
- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position



Computing the Cost Volume

AMD

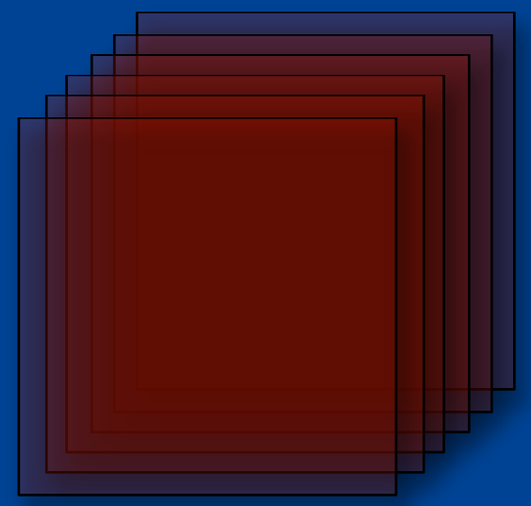
- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position



Computing the Cost Volume

AMD

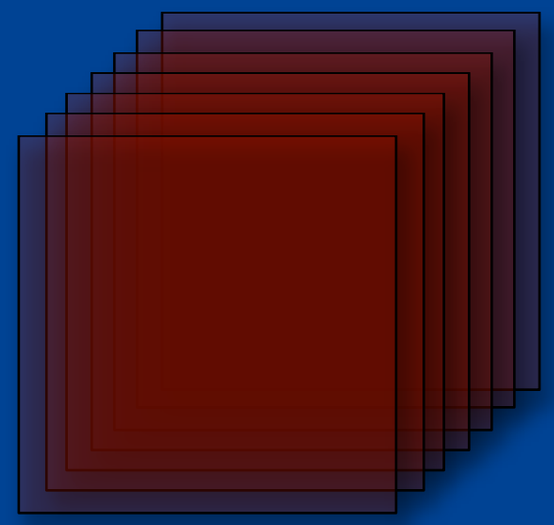
- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position



Computing the Cost Volume

AMD

- Compute volume of candidate *disparity* costs
- **Disparity:** distance between corresponding pixels the two images
- **Cost:** likelihood that two corresponding pixels are from the same real world position



Computing the Cost Volume

AMD

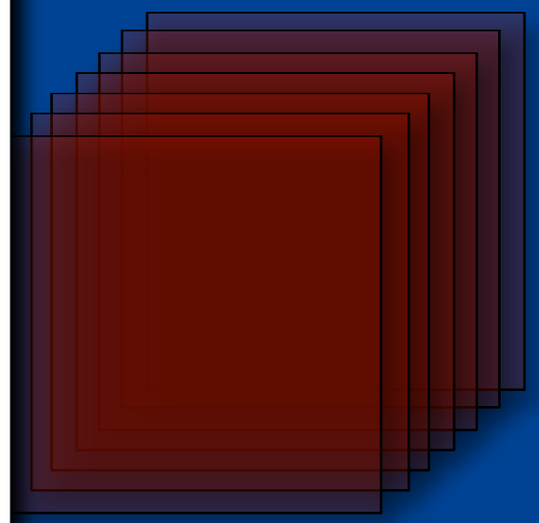
```
Output_PS ComputeCostN( float2 vpos : VPOS )
{
   float2 pos = (vpos + 0.5);

   // get the source texel (the 'p' texel)
   float3 src = tex2D( gSource, pos/gSourceSize ).rgb;

   // compute the initial offset
   pos = (pos + gStartDisparity) / gReferenceSize;

   // compute the stride
   float2 stride = gDisparityIncr / gReferenceSize;

   Output_PS outC;
   float4 dist[ NUM_OUTPUTS ];
   for( int output = 0; output < NUM_OUTPUTS; output++ )
   {
      for(int i = 0; i < 4; i++ )
      {
         // get the 'q' pixel
         float3 ref = tex2D( gReference, pos ).rgb;
         // add the stride to the position
         pos += stride;
         // compute the disparity
         (dist[output])[i] = dot( abs( src -ref ), 1.0 )/3.0;
      }
      // Set output
      outC.col[output] = dist[output];
   }
   return outC;
}
```

parity costs

esponding

nding pixels

ion



race tomorrow
SIGGRAPH2007

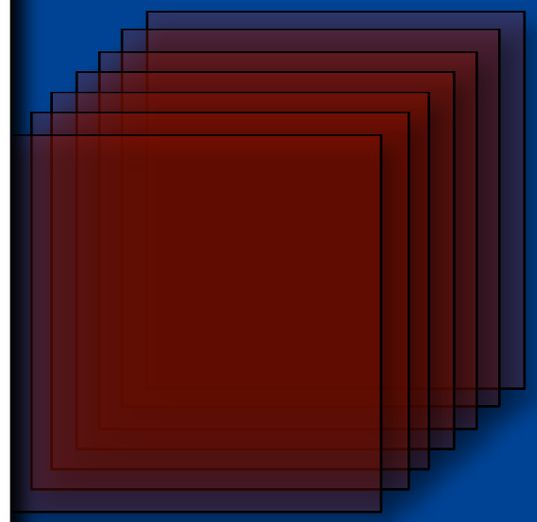
Computing the Cost Volume

parity costs

corresponding

nding pixels

```
Output_PS ComputeCostN( float2 vpos : VPOS )
{
   float2 pos = (vpos + 0.5);

   // get the source texel (the 'p' texel)
   float3 src = tex2D( gSource, pos/gSourceSize ).rgb;

   // compute the initial offset
   pos = (pos + gStartDisparity) / gReferenceSize;

   // compute the stride
   float2 stride = gDisparityIncr / gReferenceSize;

   Output_PS outC;
   float4 dist[ NUM_OUTPUTS ];
   for( int output = 0; output < NUM_OUTPUTS; output++ )
   {
      for(int i = 0; i < 4; i++ )
      {
         // get the 'q' pixel
         float3 ref = tex2D( gReference, pos ).rgb;
         // add the stride to the position
         pos += stride;
         // compute the disparity
         (dist[output])[i] = dot( abs( src -ref ), 1.0 )/3.0;
      }
      // Set output
      outC.col[output] = dist[output];
   }
   return outC;
}
```



rce tomorrow
SIGGRAPH2007

Computing the Cost Volume

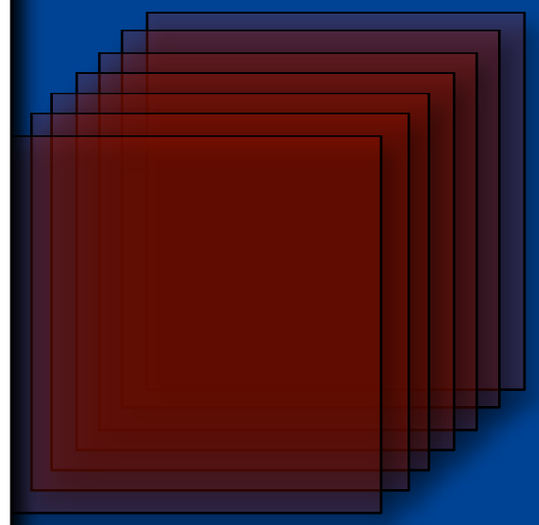
AMD

```
Output_PS ComputeCostN( float2 vpos : VPOS )
{
   float2 pos = (vpos + 0.5);

   // get the source texel (the 'p' texel)
   float3 src = tex2D( gSource, pos/gSourceSize ).rgb;

   // compute the initial offset
   pos = (pos + gStartDisparity) / gReferenceSize;

   // compute the stride
   float2 stride = gDisparityIncr / gReferenceSize;

   Output_PS outC;
   float4 dist[ NUM_OUTPUTS ];
   for( int output = 0; output < NUM_OUTPUTS; output++ )
   {
      for(int i = 0; i < 4; i++ )
      {
         // get the 'q' pixel
         float3 ref = tex2D( gReference, pos ).rgb;
         // add the stride to the position
         pos += stride;
         // compute the disparity
         (dist[output])[i] = dot( abs( src -ref ), 1.0 )/3.0;
      }
      // Set output
      outC.col[output] = dist[output];
   }
   return outC;
}
```

parity costs

corresponding

nding pixels



Computing the Cost Volume

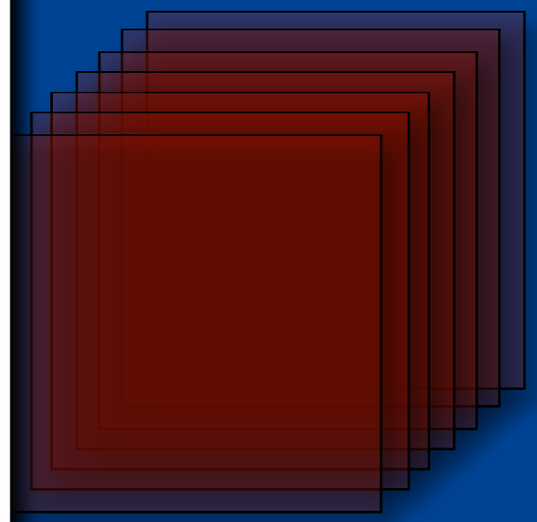
AMD

parity costs

corresponding

nding pixels

```
Output_PS ComputeCostN( float2 vpos : VPOS )
{
   float2 pos = (vpos + 0.5);

   // get the source texel (the 'p' texel)
   float3 src = tex2D( gSource, pos/gSourceSize ).rgb;

   // compute the initial offset
   pos = (pos + gStartDisparity) / gReferenceSize;

   // compute the stride
   float2 stride = gDisparityIncr / gReferenceSize;

   Output_PS outC;
   float4 dist[ NUM_OUTPUTS ];
   for( int output = 0; output < NUM_OUTPUTS; output++ )
   {
      for(int i = 0; i < 4; i++ )
      {
         // get the 'q' pixel
         float3 ref = tex2D( gReference, pos ).rgb;
         // add the stride to the position
         pos += stride;
         // compute the cost for this disparity
         (dist[output])[i] = dot( abs( src -ref ), 1.0 )/3.0;
      }
      // Set output
      outC.col[output] = dist[output];
   }
   return outC;
}
```



race tomorrow
SIGGRAPH2007

Aggregate Cost Volume

AMD

Aggregate Cost Volume

- Update cost volume
  - Based on neighborhood around reference point
  - Color and Euclidean distance

Aggregate Cost Volume

- Update cost volume
  - Based on neighborhood around reference point
  - Color and Euclidean distance
- Approximate 2D neighborhood with two 1D kernels
  - Filter kernel not separable
  - Not mathematically correct
  - Does a good enough job

Aggregate Cost Volume

AMD

- Update cost volume
  - Based on neighborhood around reference point
  - Color and Euclidean distance
- Approximate 2D neighborhood with two 1D kernels
  - Filter kernel not separable
  - Not mathematically correct
  - Does a good enough job
- See [Wang06] for full details

Find Best Depth Estimate

- Step through cost volume
- Find disparity with minimum cost
  - "Winner-takes-all"
- Compute depth from "best" disparity

Find Best Depth Estimate

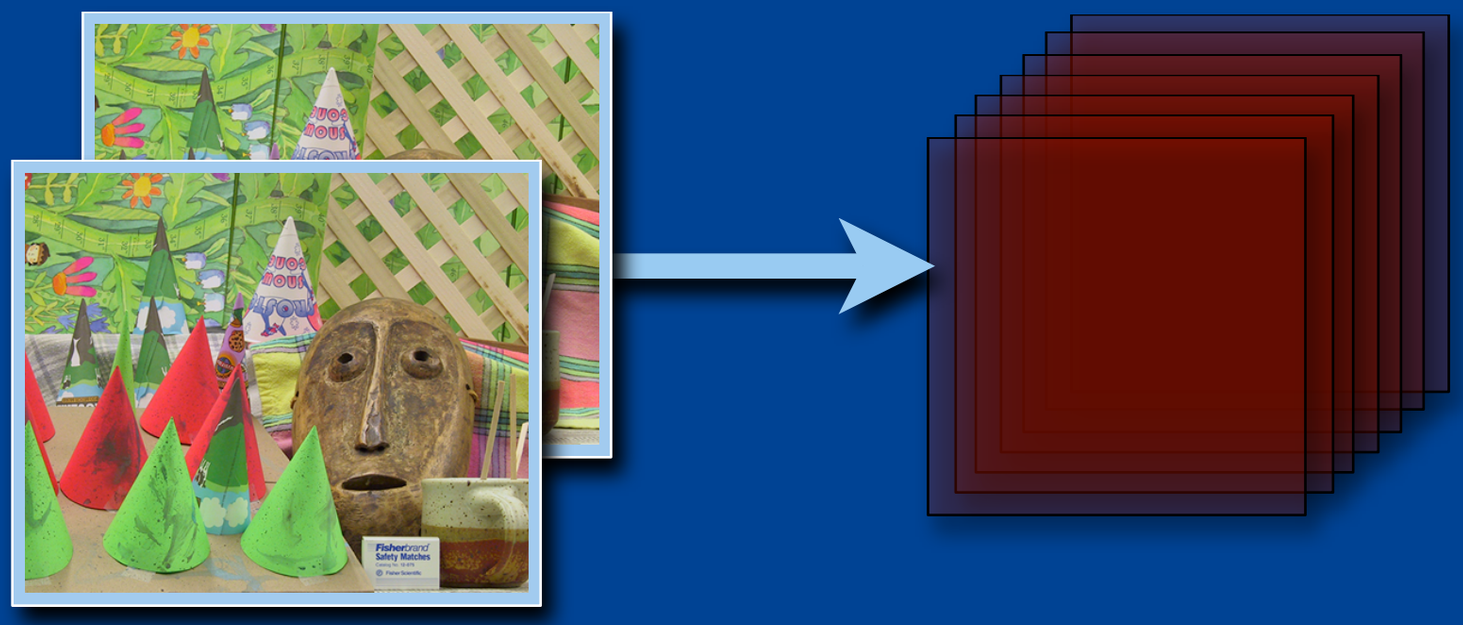
- Step through cost volume
- Find disparity with minimum cost
  - “Winner-takes-all”
- Compute depth from “best” disparity

Find Best Depth Estimate

- Step through cost volume
- Find disparity with minimum cost
  - “Winner-takes-all”
- Compute depth from “best” disparity

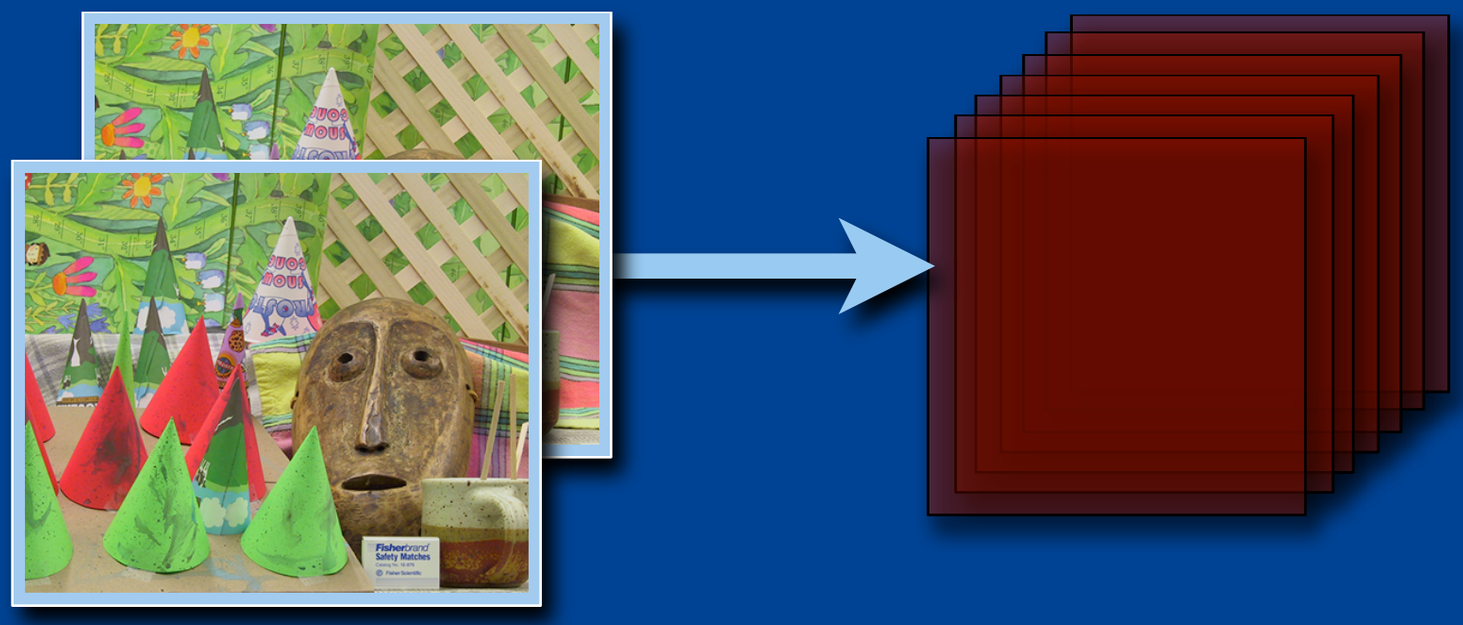
Find Best Depth Estimate

- Step through cost volume
- Find disparity with minimum cost
  - “Winner-takes-all”
- Compute depth from “best” disparity



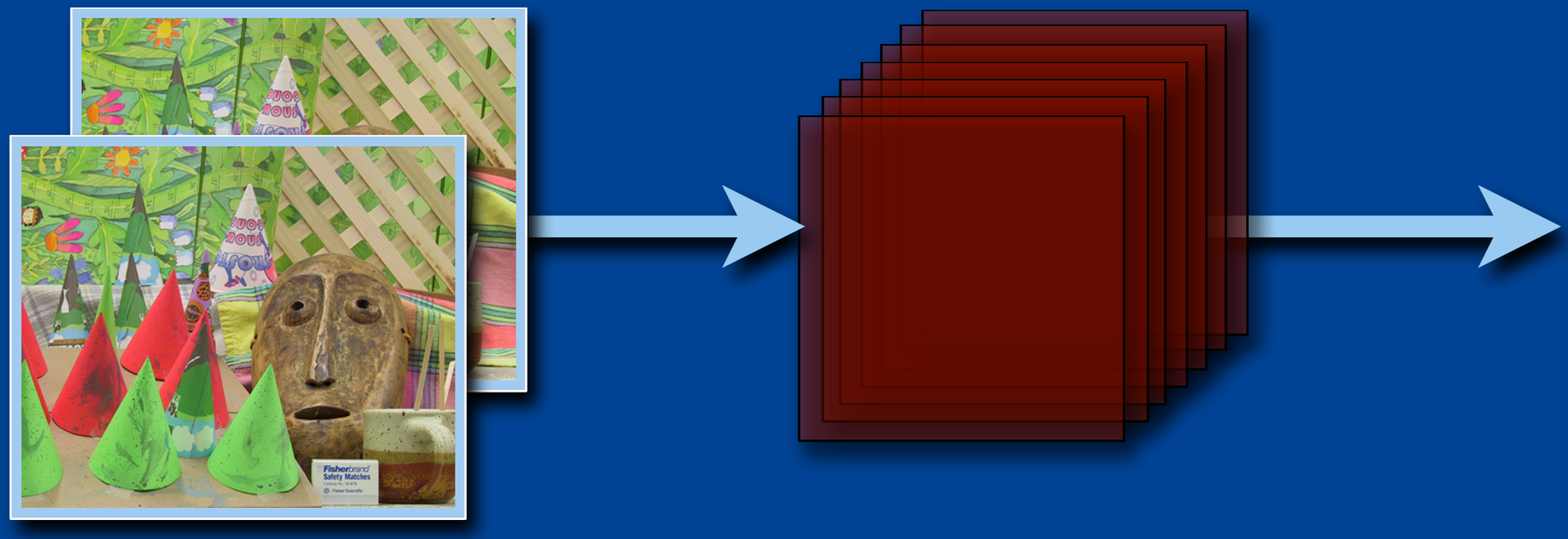
Find Best Depth Estimate

- Step through cost volume
- Find disparity with minimum cost
  - "Winner-takes-all"
- Compute depth from "best" disparity



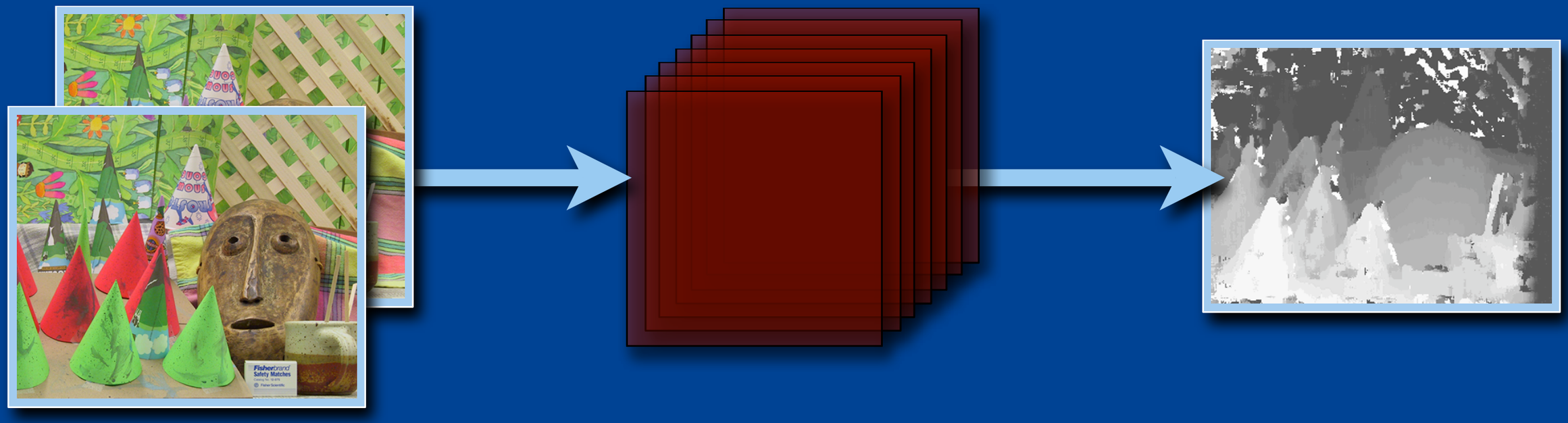
Find Best Depth Estimate

- Step through cost volume
- Find disparity with minimum cost
  - “Winner-takes-all”
- Compute depth from “best” disparity



Find Best Depth Estimate

- Step through cost volume
- Find disparity with minimum cost
  - “Winner-takes-all”
- Compute depth from “best” disparity



Physics Implementation
(in brief)

SIGGRAPH2007

Physics Implementation

- Leverages both the CPU and GPU
- Quick physics refresher:

Physics Implementation

- Leverages both the CPU and GPU
- Quick physics refresher:

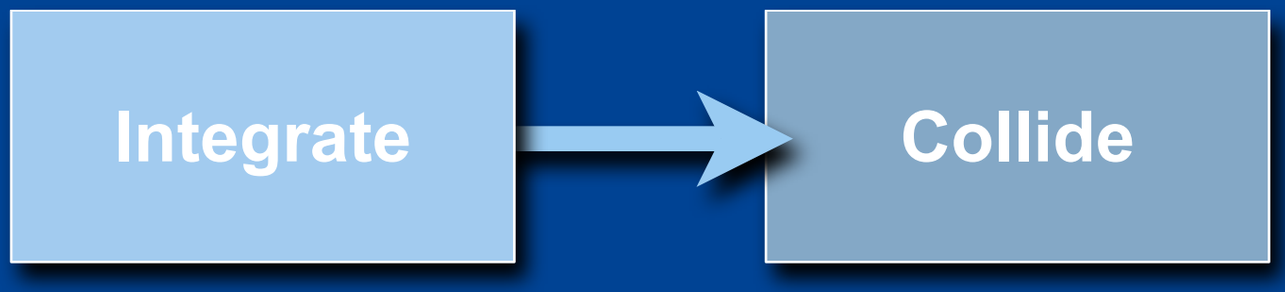
Integrate

Physics Implementation

- Leverages both the CPU and GPU
- Quick physics refresher:

Physics Implementation

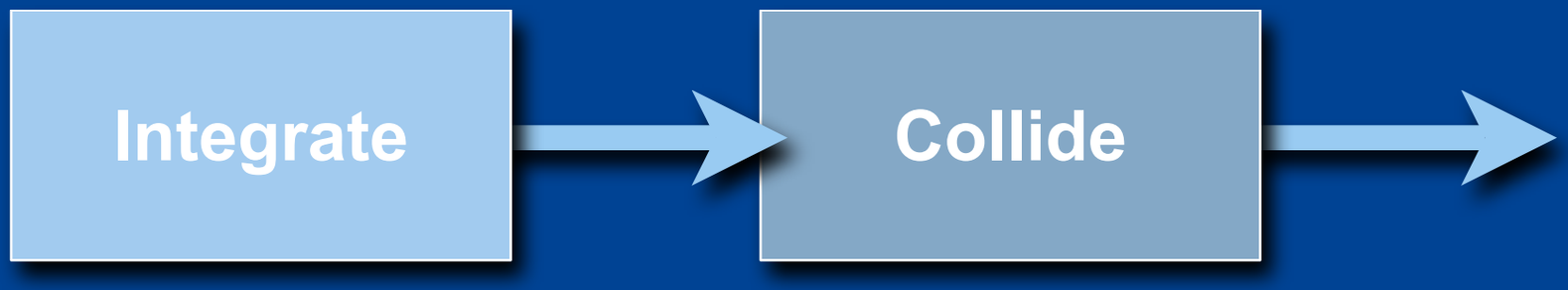
- Leverages both the CPU and GPU
- Quick physics refresher:



face tomorrow
SIGGRAPH2007

Physics Implementation

- Leverages both the CPU and GPU
- Quick physics refresher:



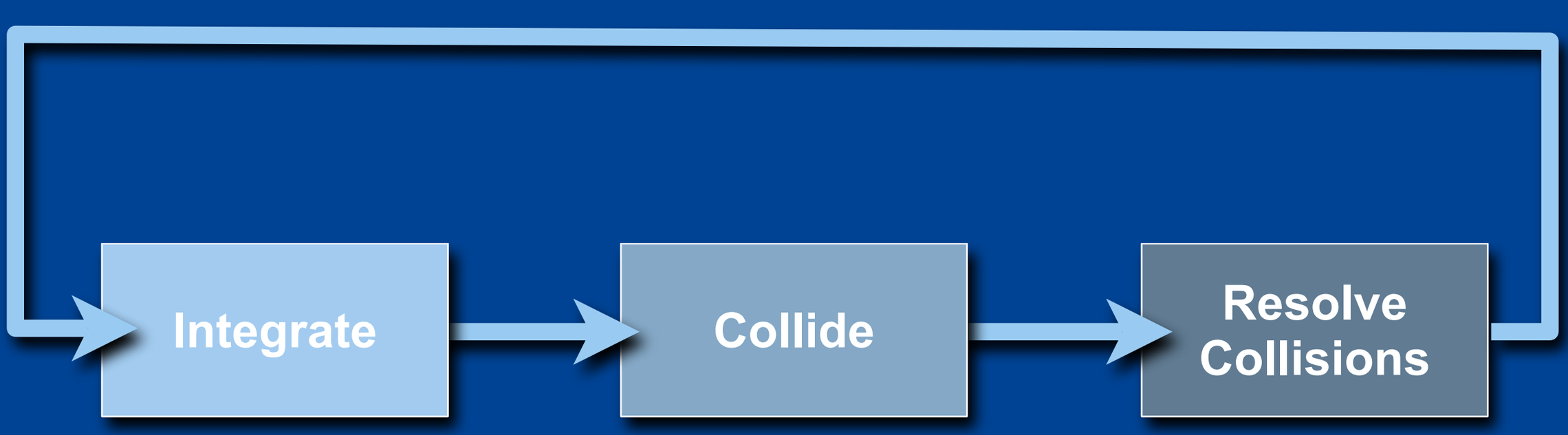
face tomorrow
SIGGRAPH2007

Physics Implementation

- Leverages both the CPU and GPU
- Quick physics refresher:

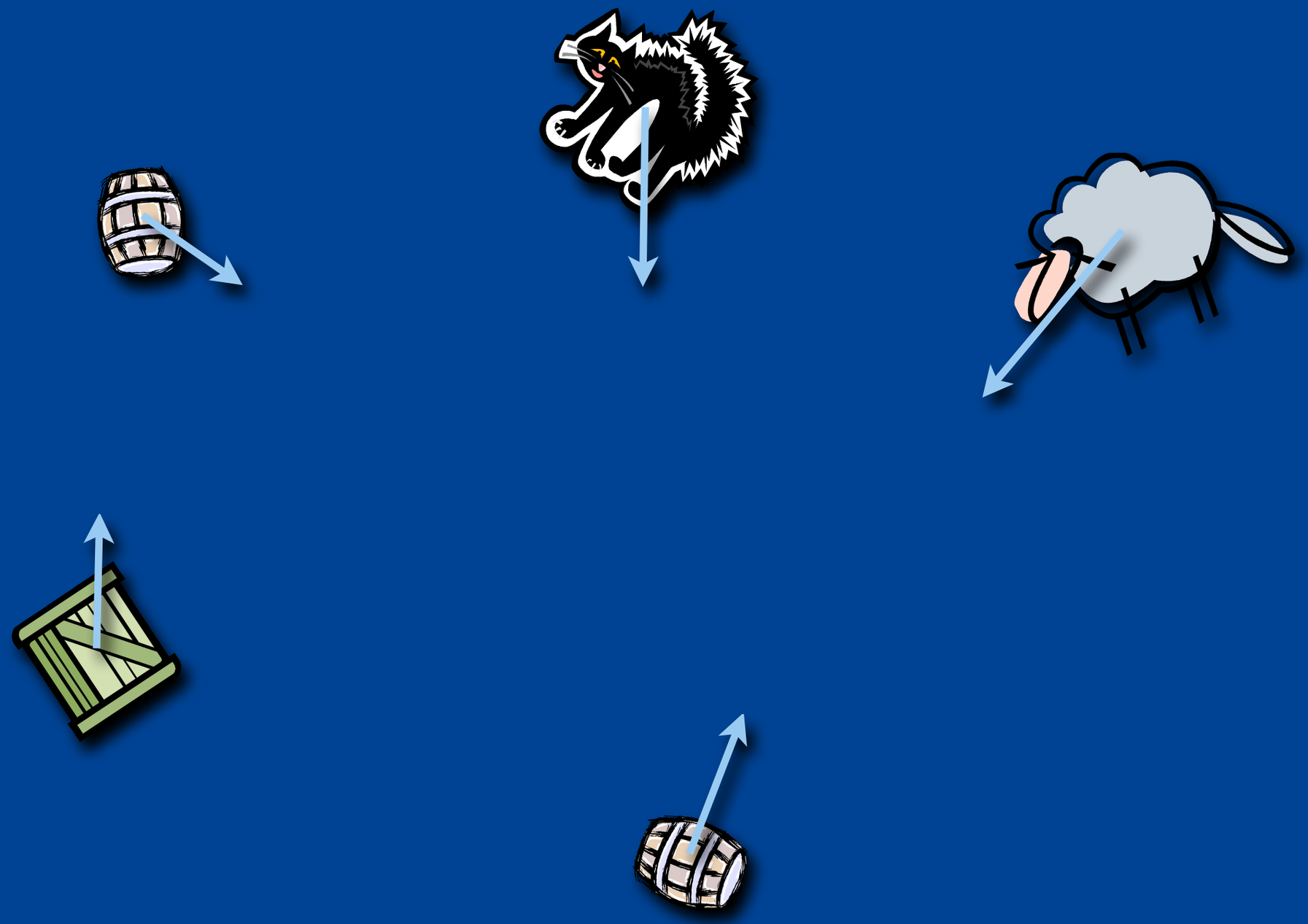
Physics Implementation

- Leverages both the CPU and GPU
- Quick physics refresher:



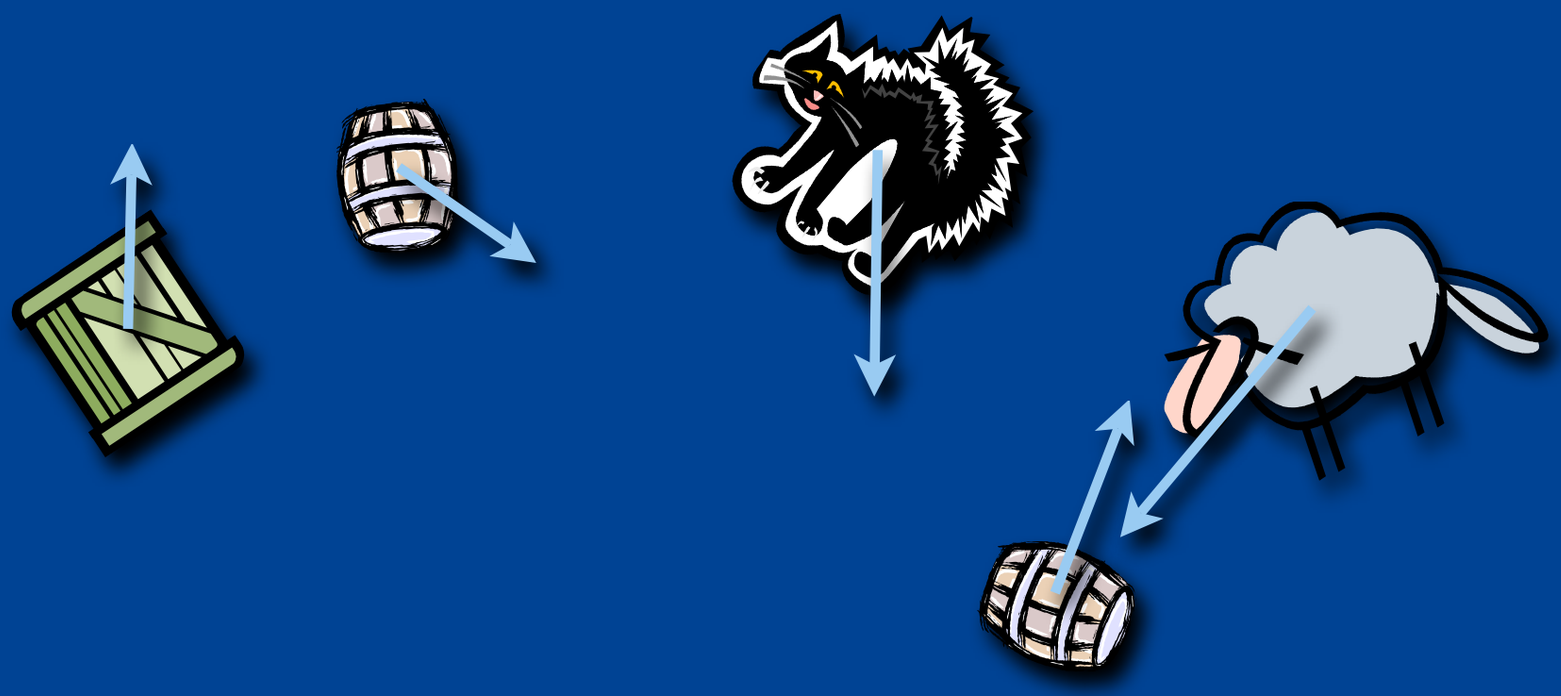
face tomorrow
SIGGRAPH2007

Integration



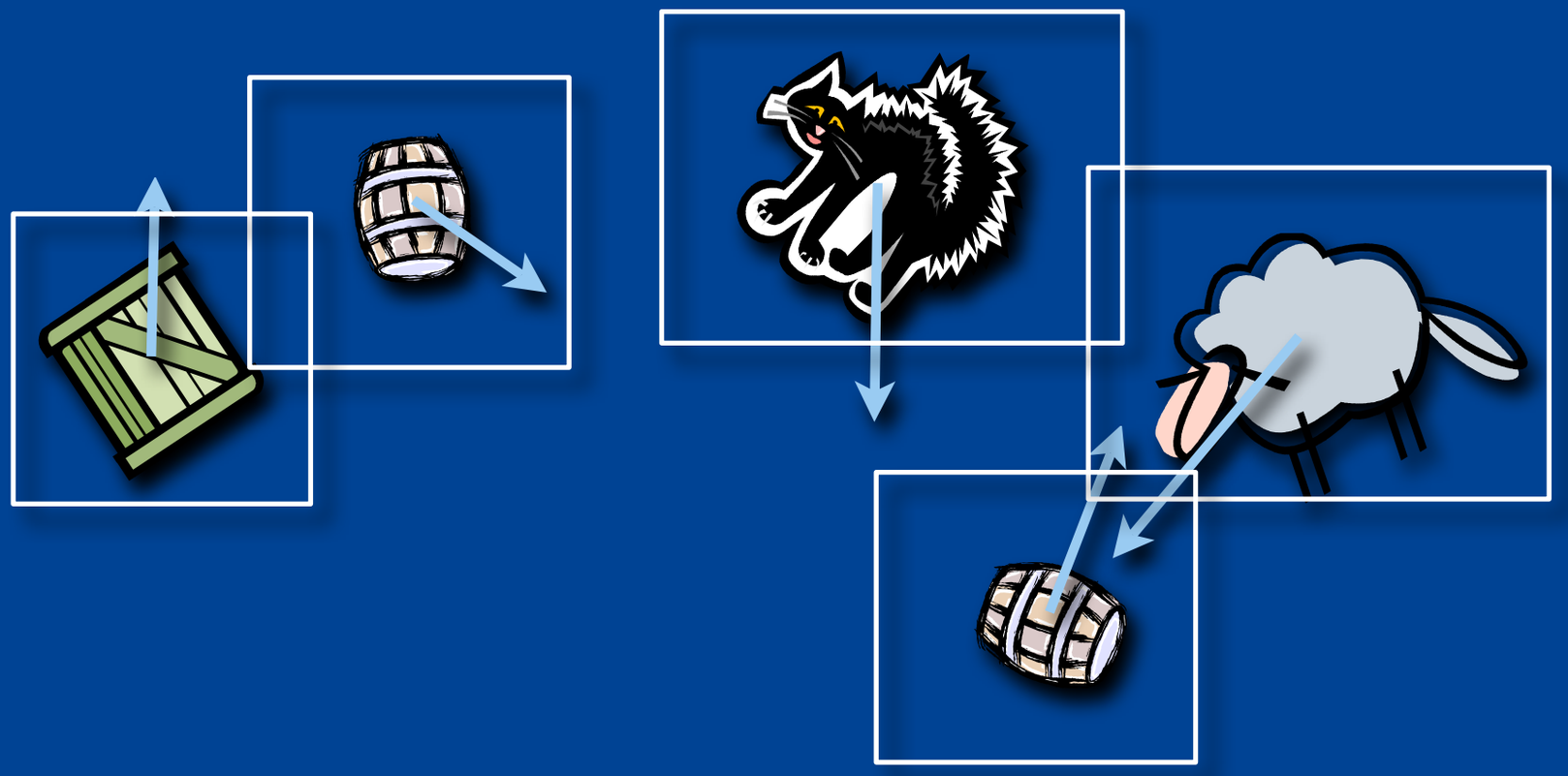
Integration

AMD



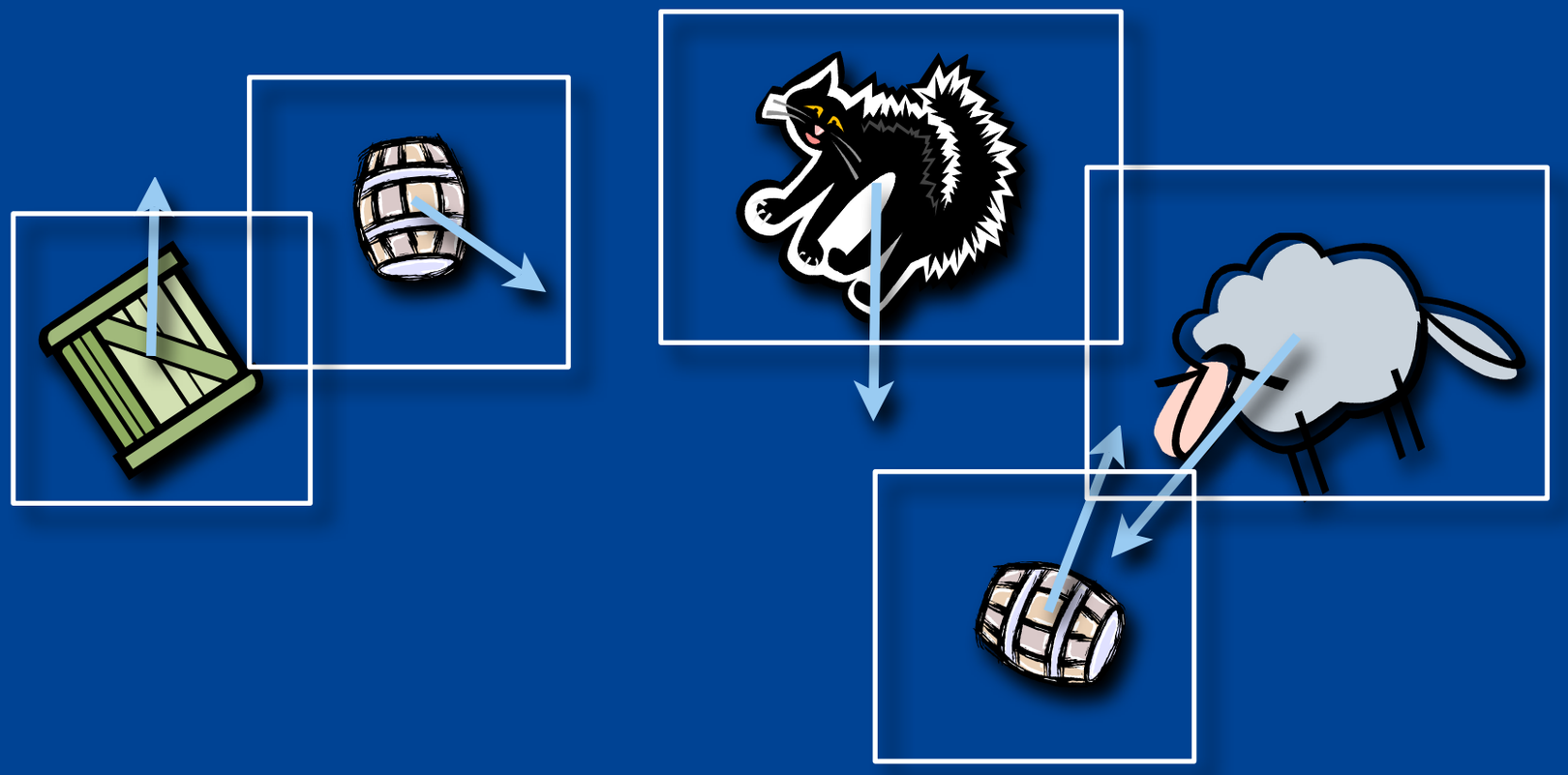
Integration

AMD



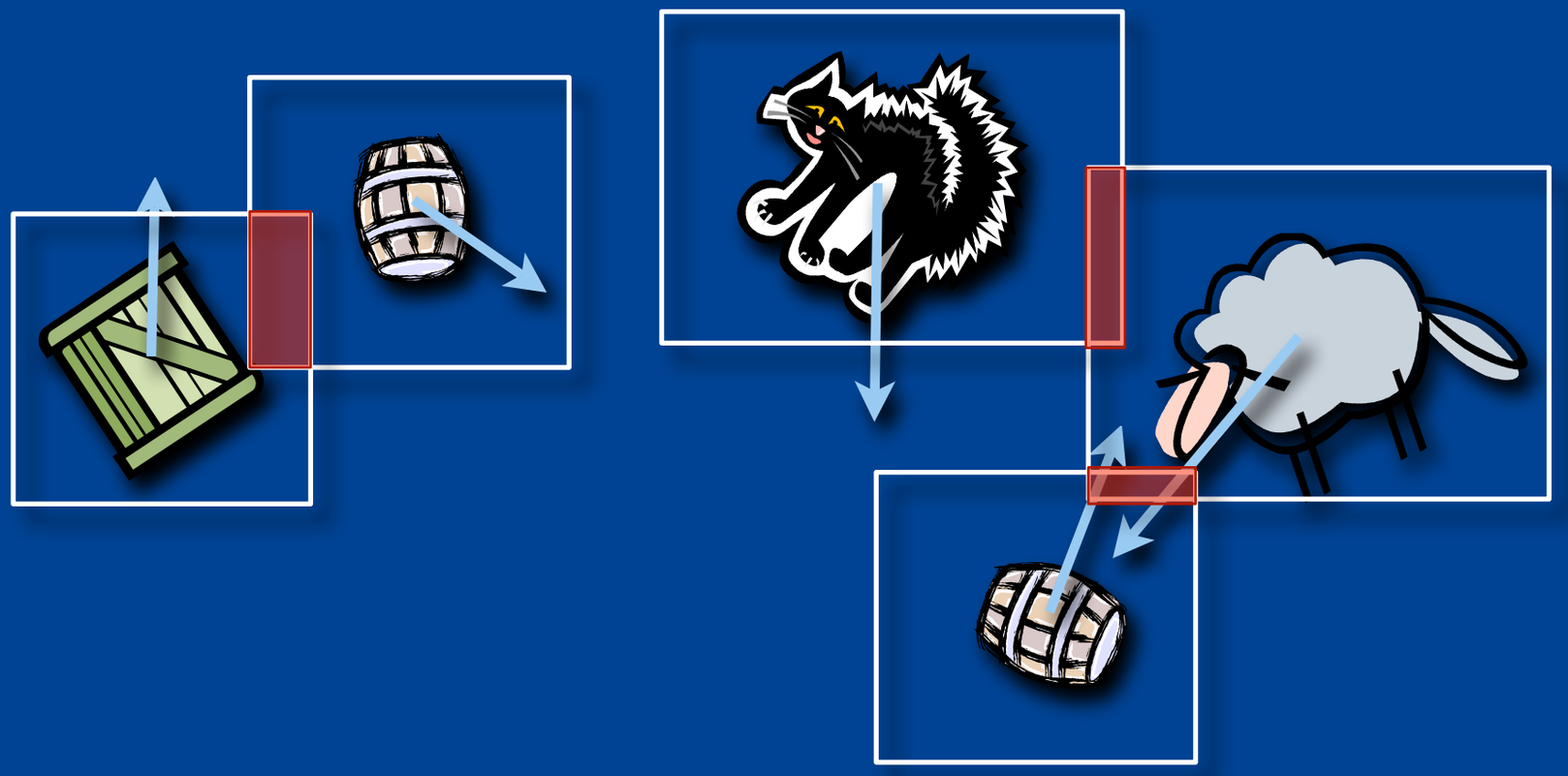
Find Overlapping Pairs

AMD

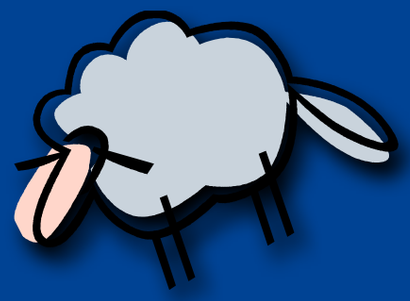


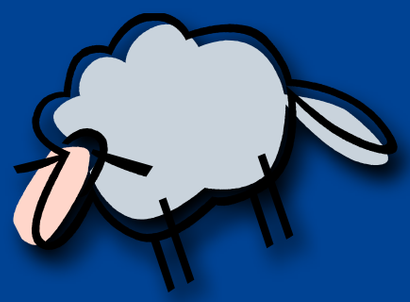
Find Overlapping Pairs

AMD



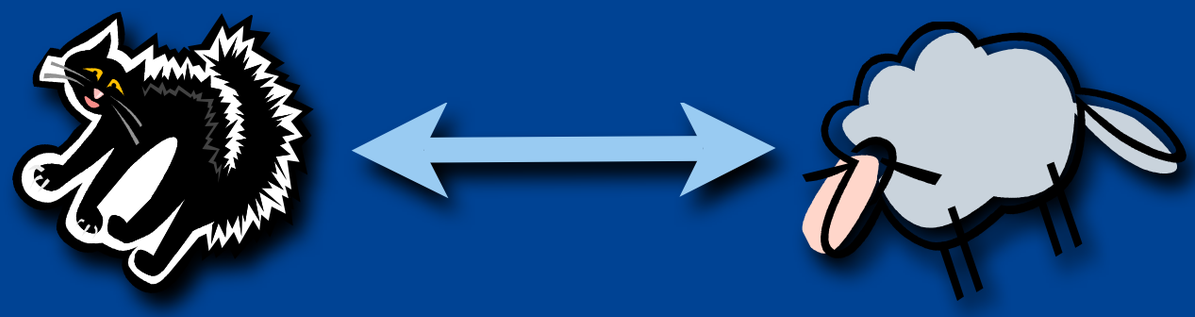
Narrow Phase Collide

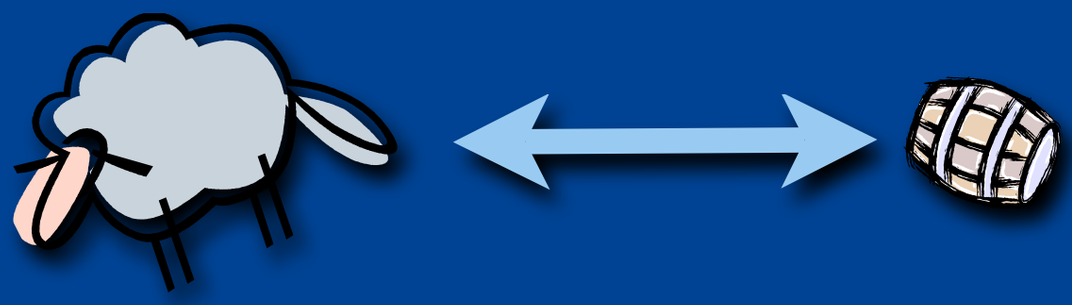


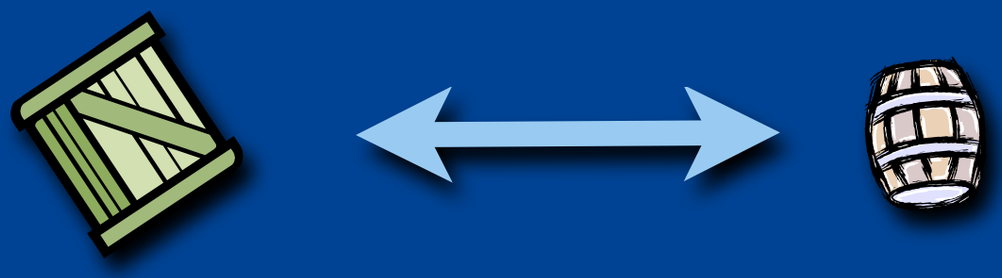


Narrow Phase Collide

AMD





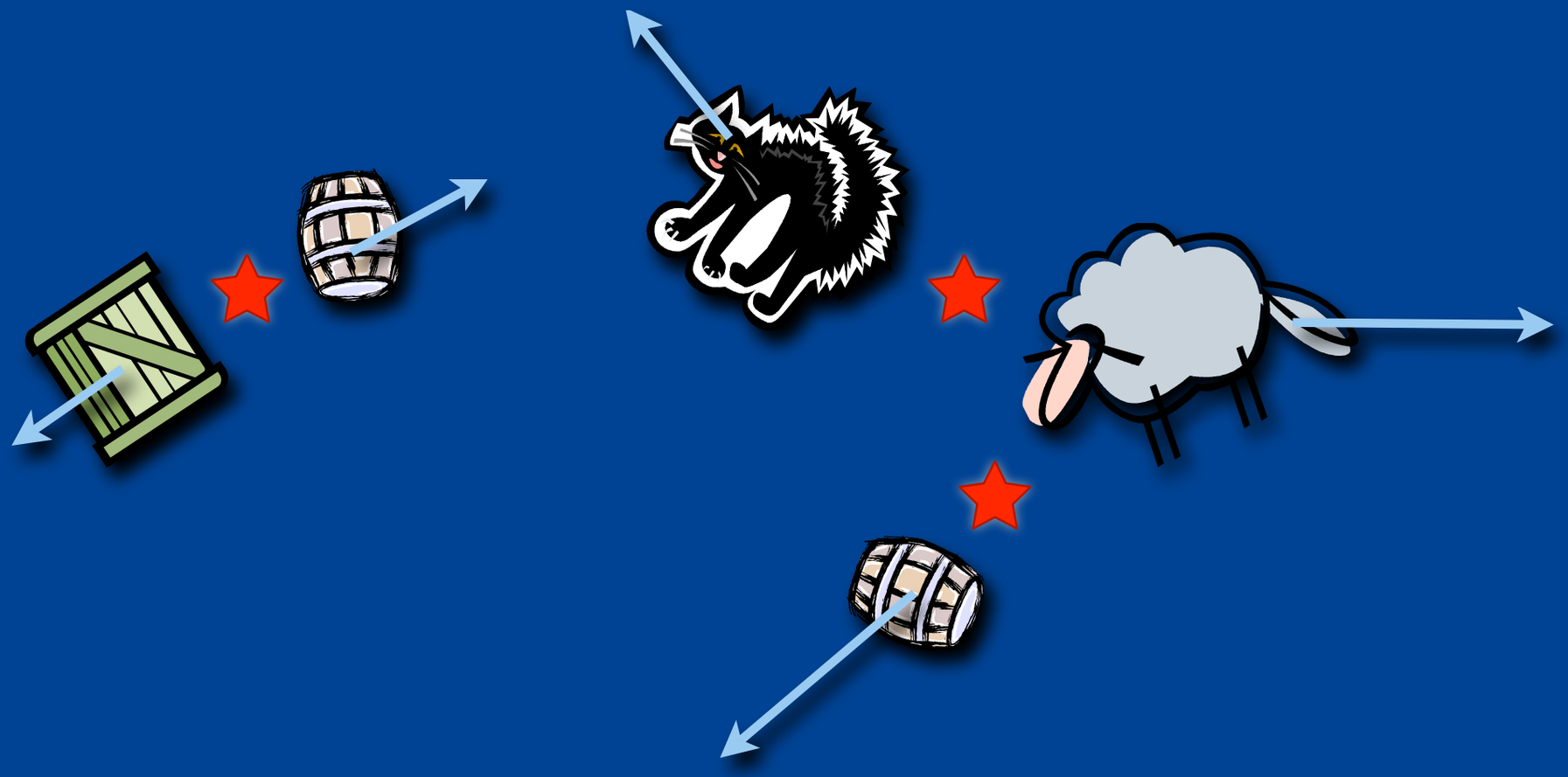


Solve Collisions

AMD

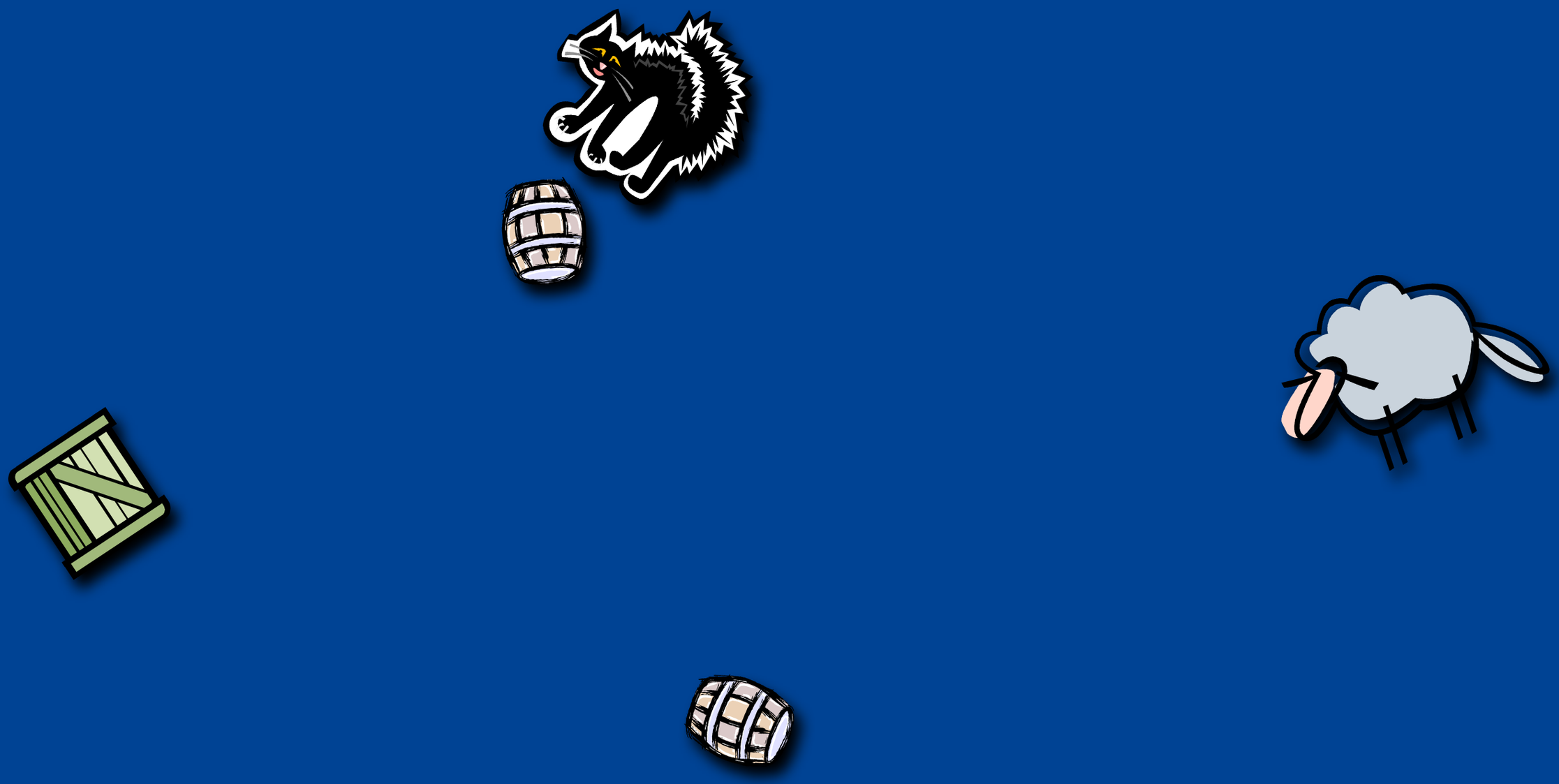
Solve Collisions

AMD



Solve Collisions

AMD



Parallelism of Physics Sim

AMD

- Integrate
  - *“100%”* parallel - easy to use GPU
- Collide
  - *“70%”* parallel - use both CPU & GPU
  - “Broad phase” on the CPU
  - “Narrow phase” on the GPU
- Resolve
  - *“99%”* parallel - GPU (possibly multiple passes)

Putting it together

Putting it together

- Capture image pair from camera

Putting it together

AMD

- Capture image pair from camera
- Rectify images and extract depth
  - GPU only

Putting it together

- Capture image pair from camera
- Rectify images and extract depth
  - GPU only
- Generate model from depth
  - GPU only

Putting it together

AMD

- Capture image pair from camera
- Rectify images and extract depth
  - GPU only
- Generate model from depth
  - GPU only
- Run rigid-body physics simulation
  - Both GPU-CPU

Putting it together

AMD

- Capture image pair from camera
- Rectify images and extract depth
  - GPU only
- Generate model from depth
  - GPU only
- Run rigid-body physics simulation
  - Both GPU-CPU
- Update and render scene

Putting it together

AMD

- Capture image pair from camera
- Rectify images and extract depth
  - GPU only
- Generate model from depth
  - GPU only
- Run rigid-body physics simulation
  - Both GPU-CPU
- Update and render scene
- Rinse and repeat

Demo

AMD

Conclusion and Questions

AMD

- Leverage GPU to enable novel input paradigms
- Keep data on GPU as much as possible
- Utilize CTM for better programmability of GPU
  - CTM info: **streamcomputing@amd.com**
- Point Grey Bumblebee2 camera used for input

References

[wang06] Wang, L., Liao, M., Gong, M., Yang, R., and Nister, D. 2006. High-Quality Real-Time Stereo Using Adaptive Cost Aggregation and Dynamic Programming. 3DPVT'06

Advanced Human Computer Interface

Demo

SIGGRAPH2007