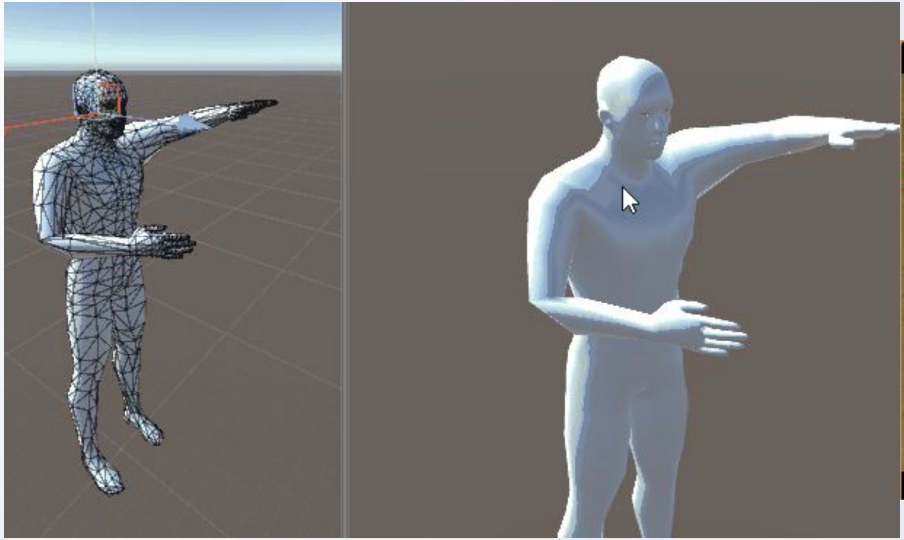
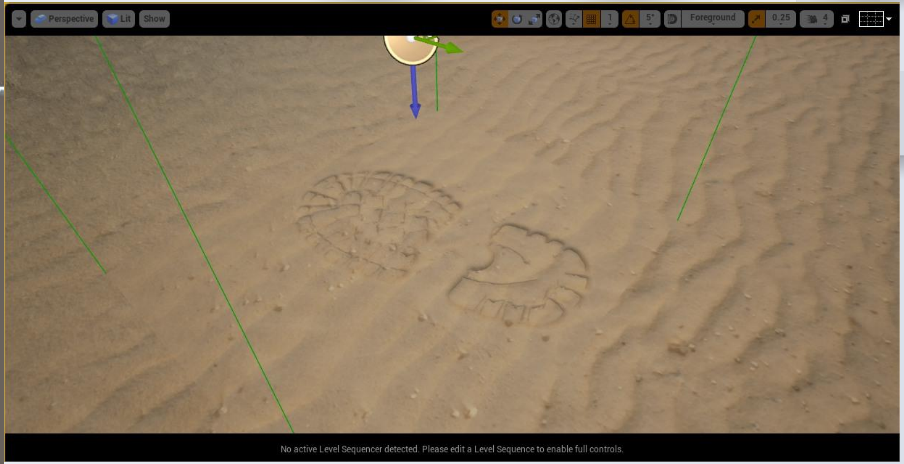
DECALS
BILLBOARDS
DEPTH OF FIELD
MOTION BLUR

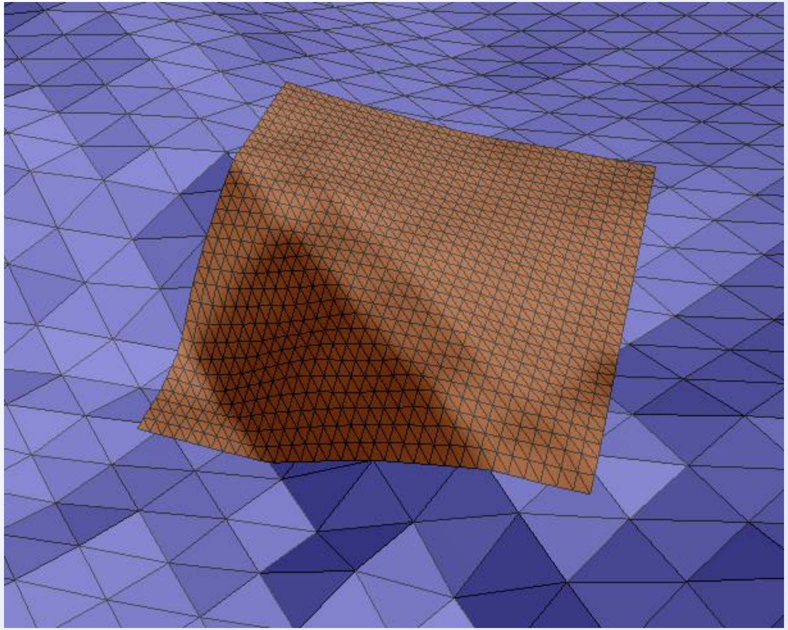
Art Parkeenvincha | Terence So

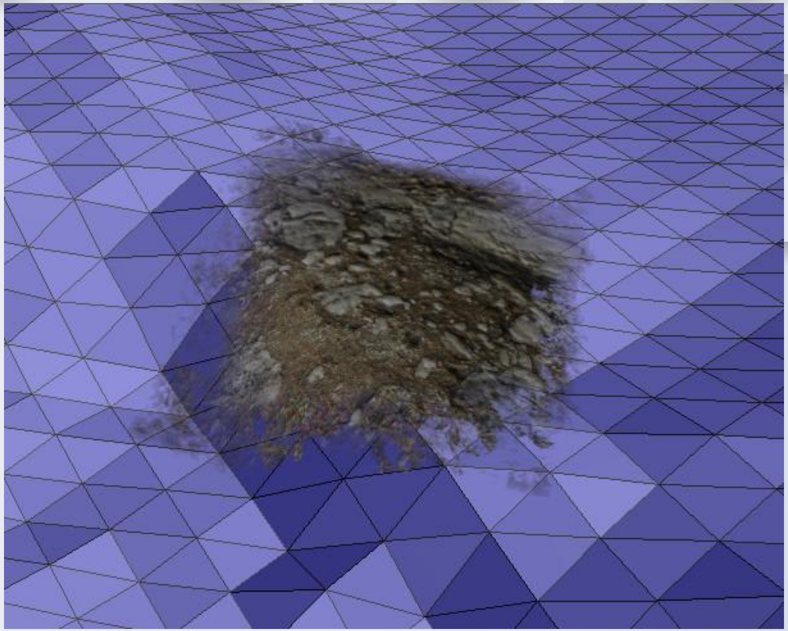
DECALS





Examples of decals

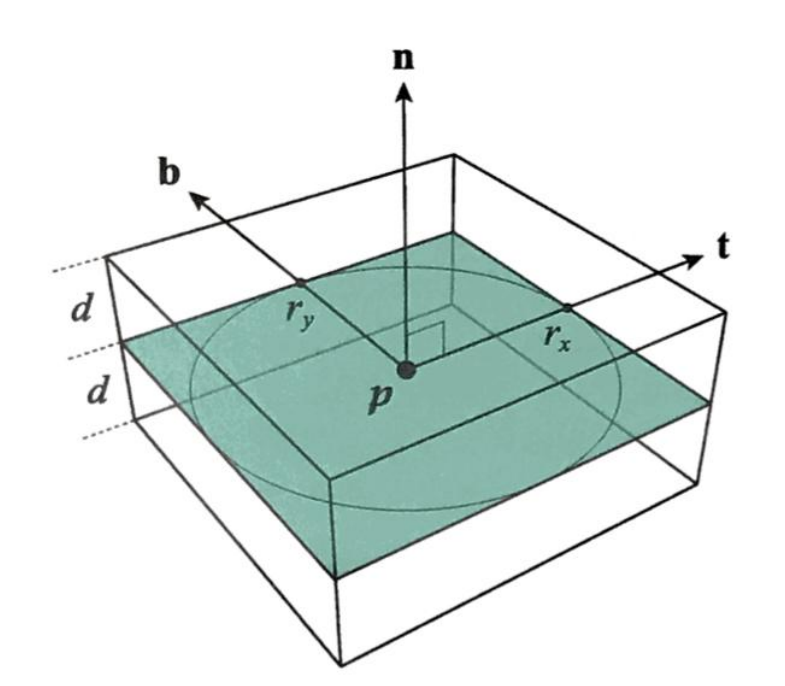




Examples of decals

How does it work?

1. Get center of decal, **p**
2. Get normal of geometry, **n**
3. Compute tangent vector, **t**
4. Compute bitangent vector, **b**
5. Compute bounding box with distance **d**.
6. Compute **decal clipping**.



Focus on step #5 and #6

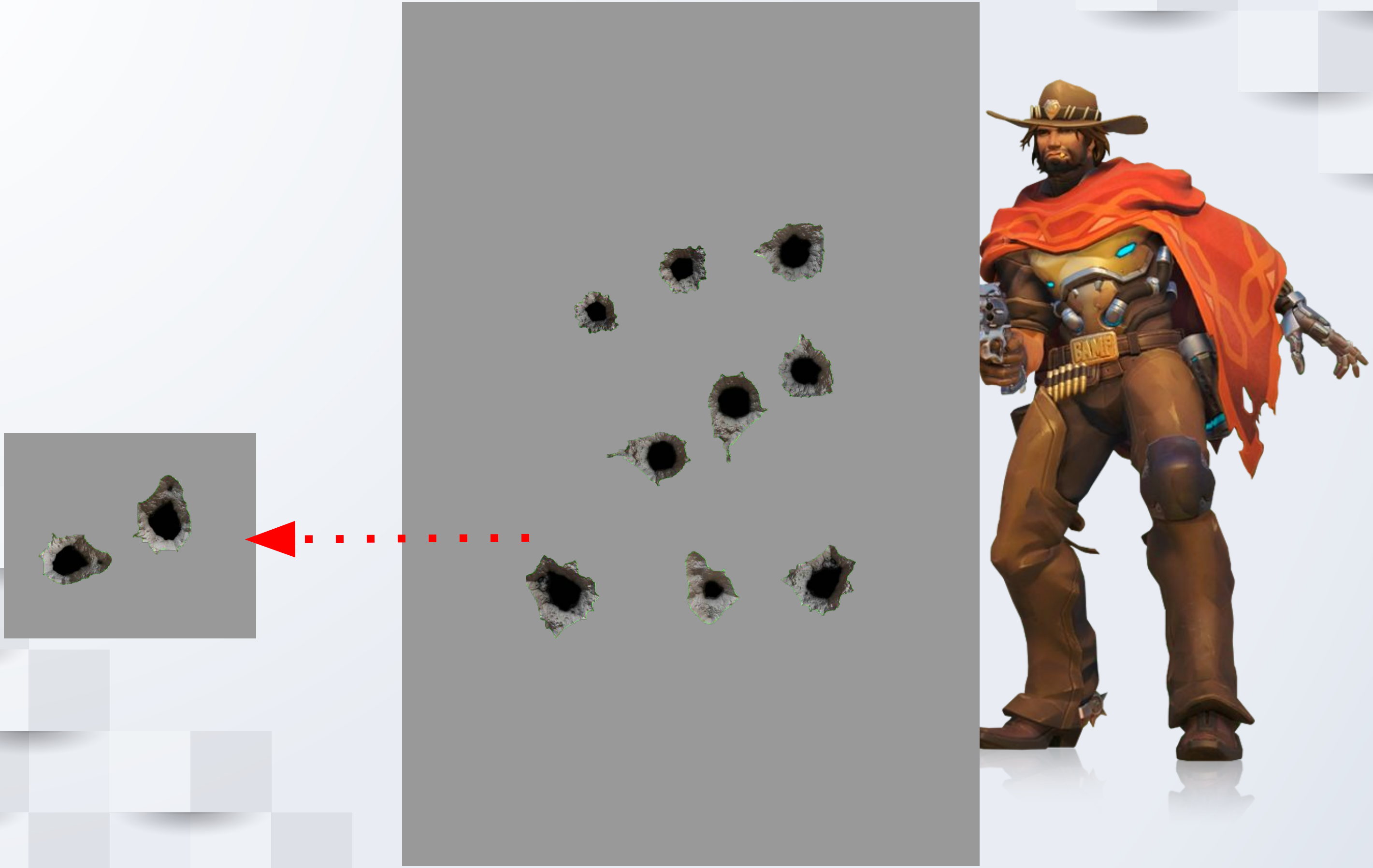
Compute bounding box with distance **d**.

Compute **decal clipping**.

Focus on step #5 and #6

Compute bounding box with distance **d**.

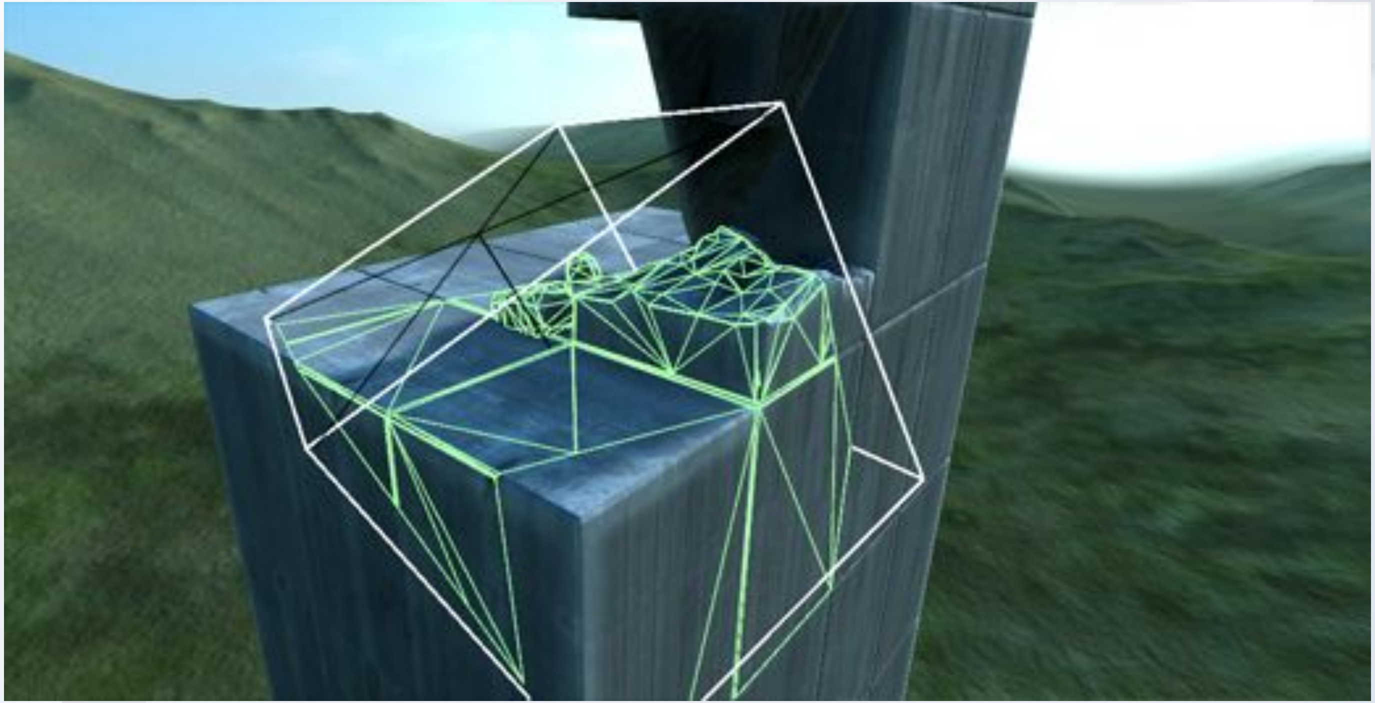
Compute **decal clipping**.

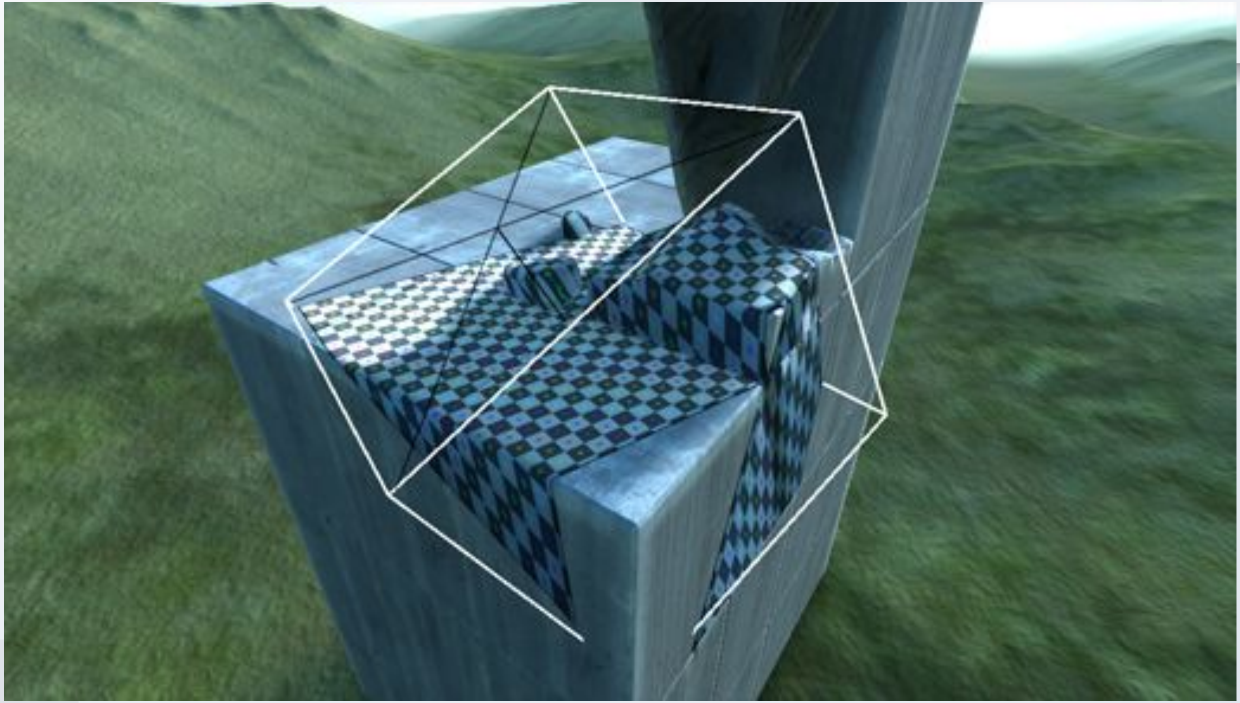


Focus on step #5 and #6

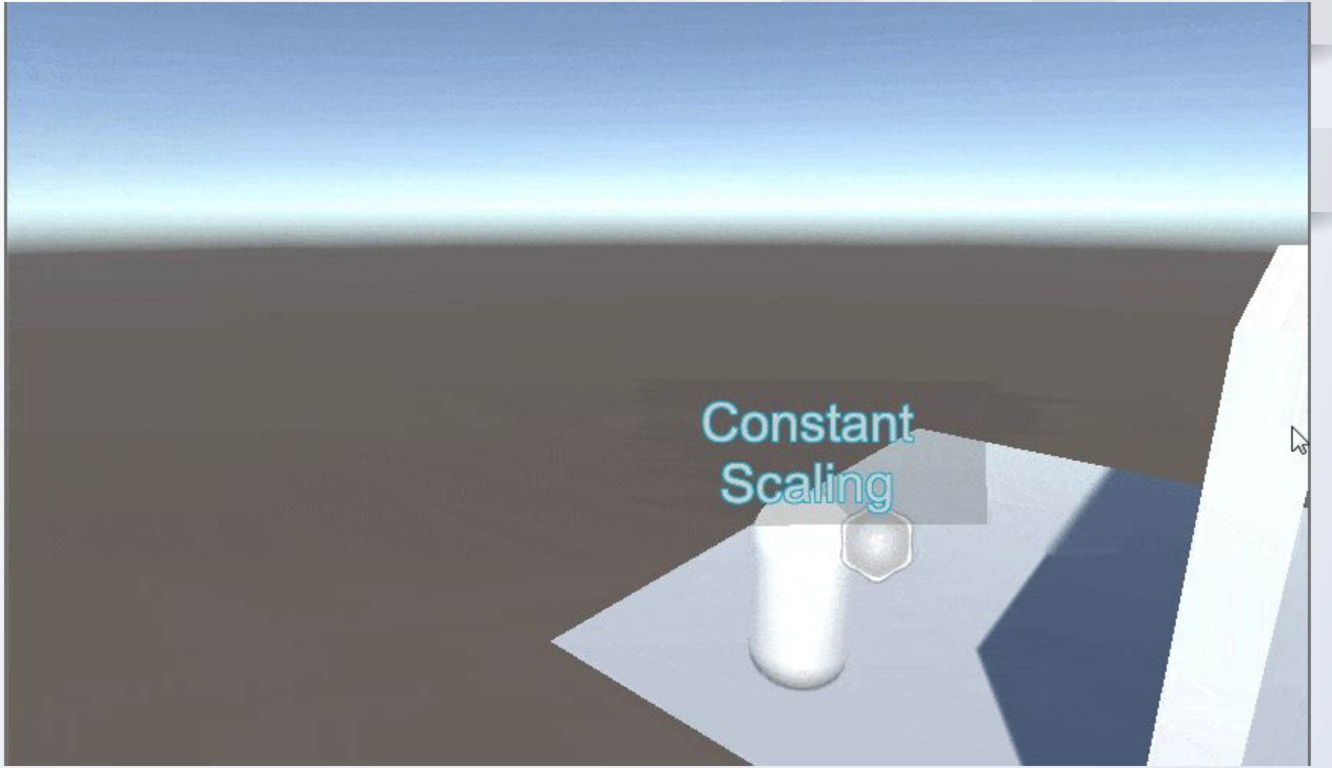
Compute bounding box with distance **d**.

Compute **decal clipping**.

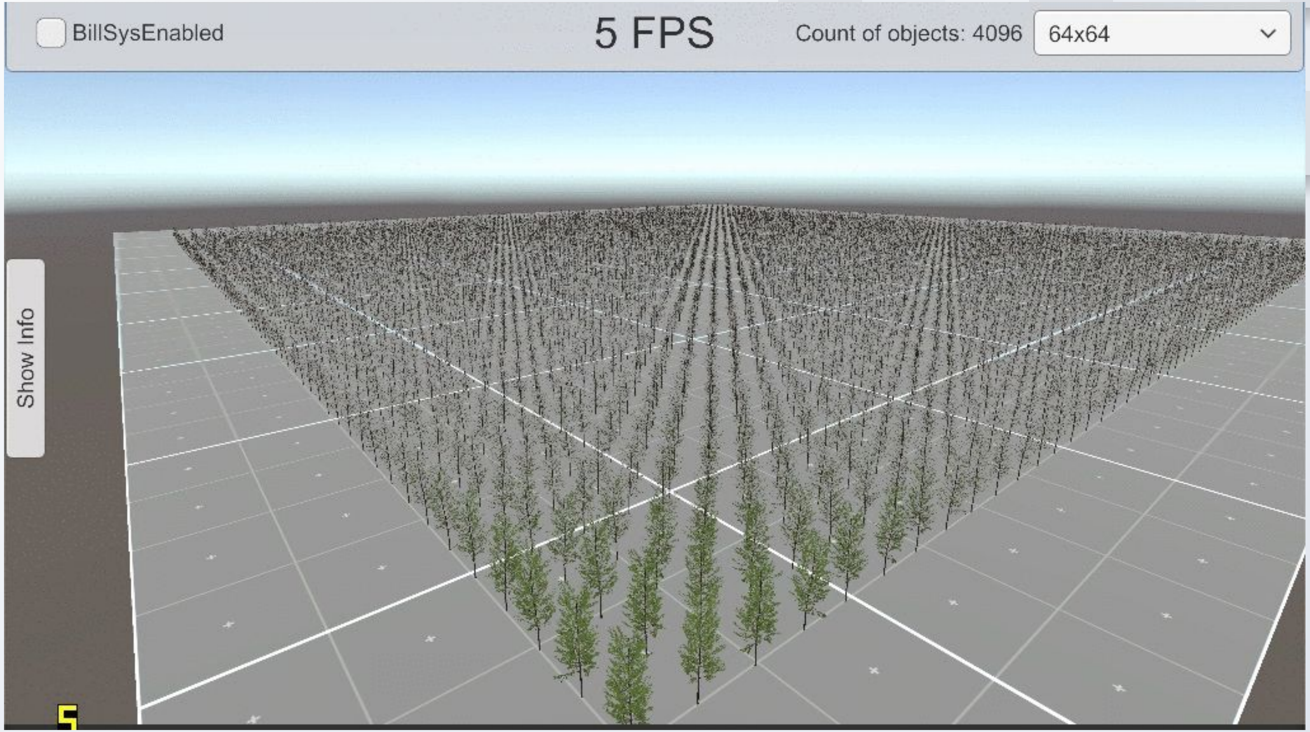




BILLBOARDS

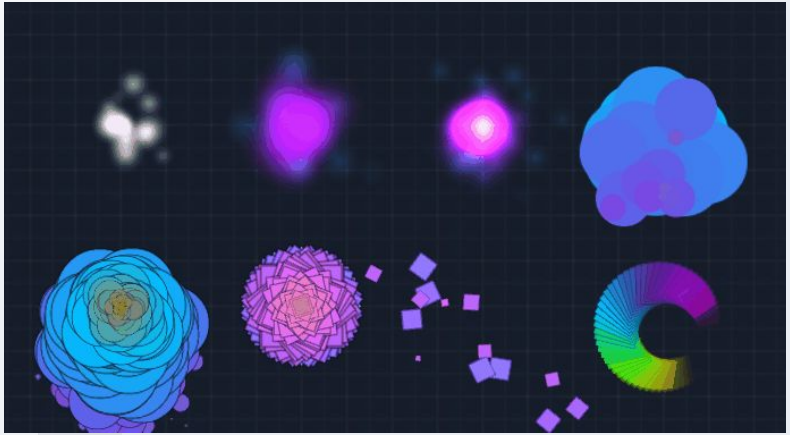


Examples of billboards

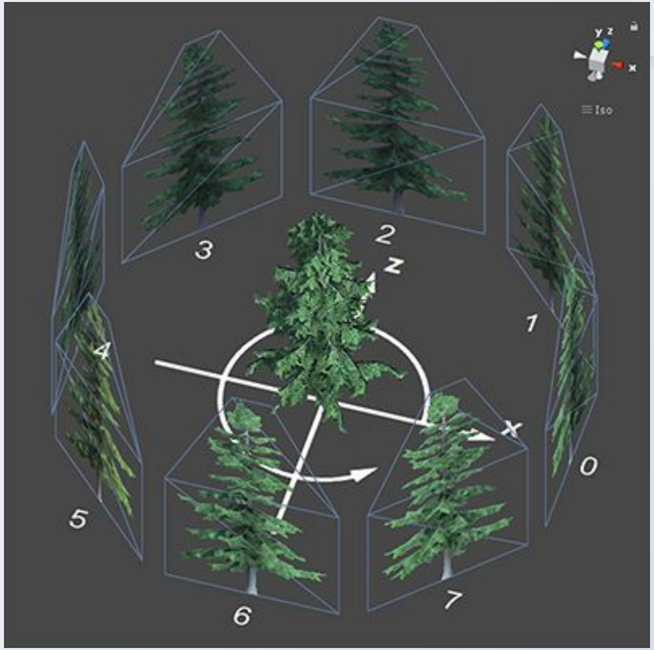


Examples of billboards (impostors)

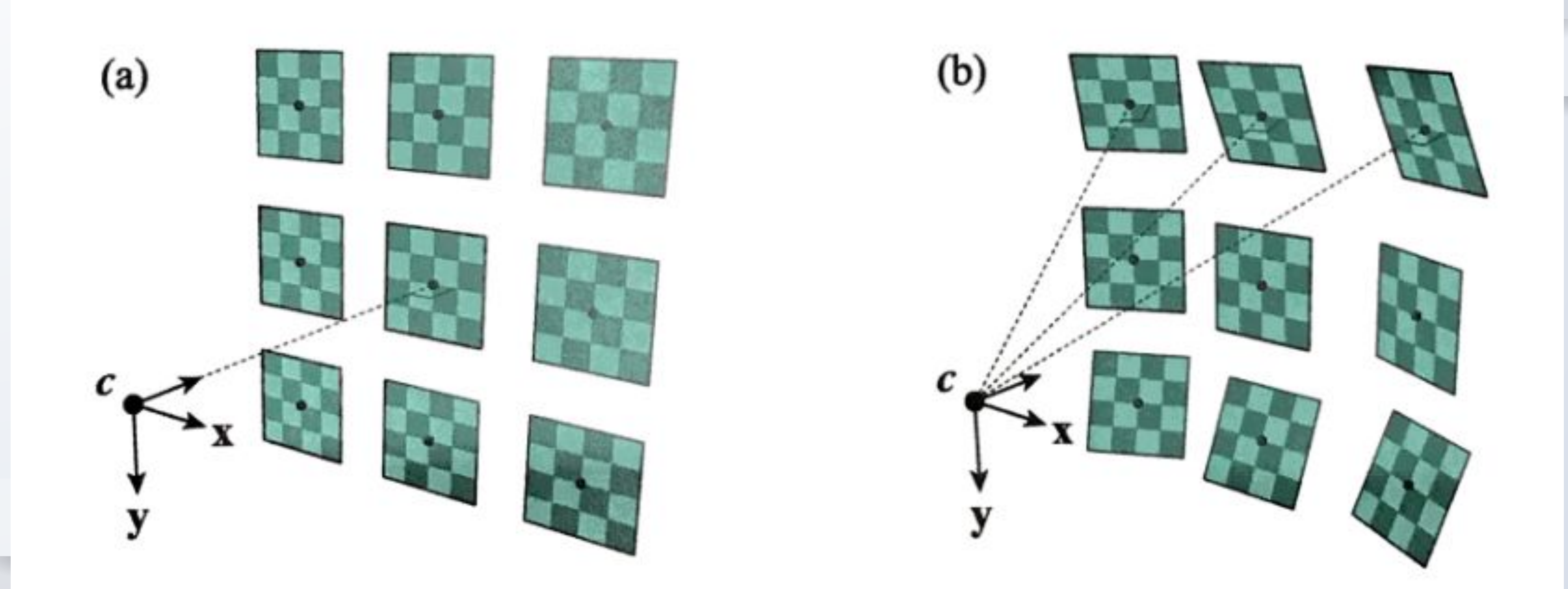
Spherical



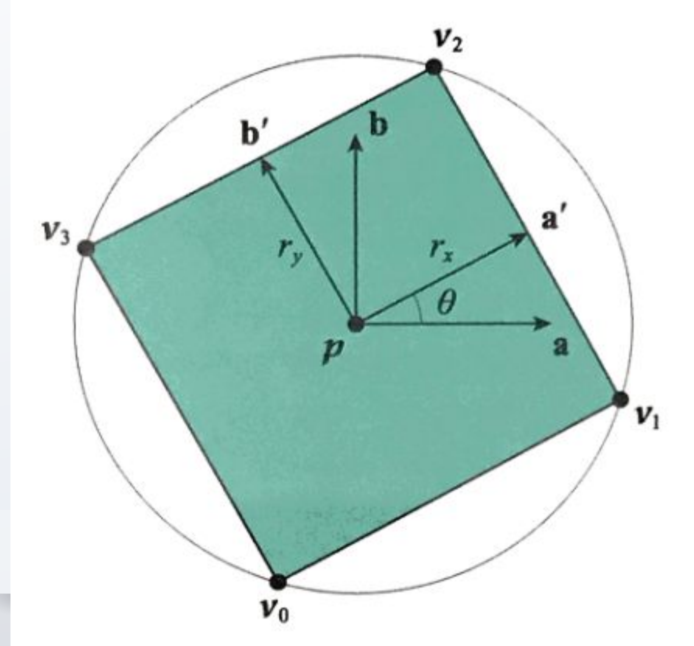
Cylindrical



Spherical billboards



Spherical billboards

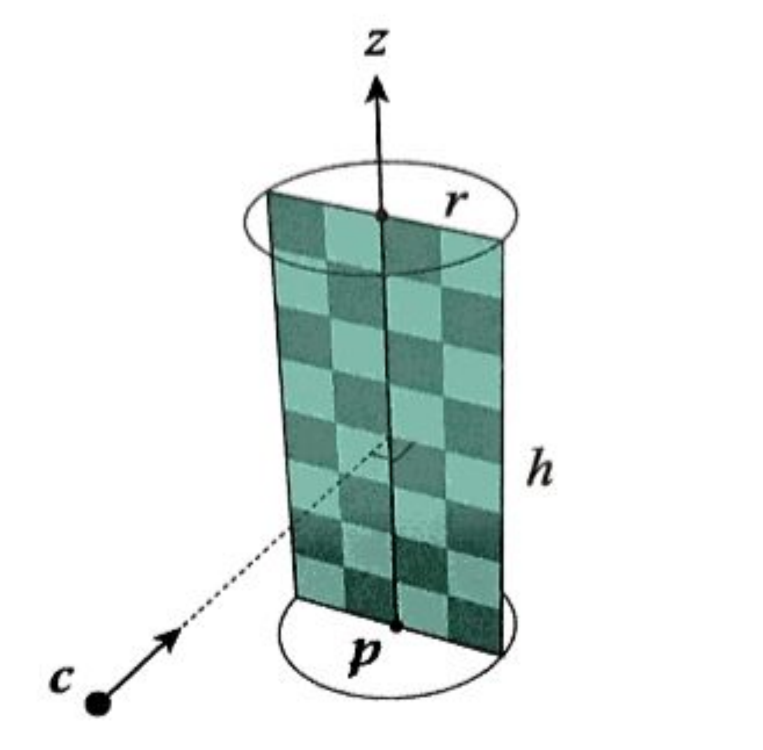


1. Get camera position, **c**.
2. Set billboard's normal, **n**, to face towards **c**.
3. Calculate tangent vector, **a**.
4. Calculate another tangent vector, **b**.
5. Compute all **4 vertices** of the billboard.

Cylindrical billboards

Computed the same way as a spherical billboard.

Except the tangent vector **a** needs to be perpendicular to the z-axis.



MOTION BLUR

1. **What is it?**

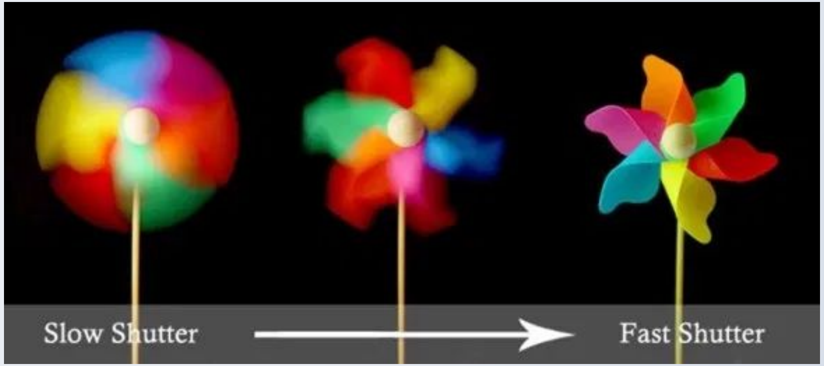
2. **Implementation Methods**

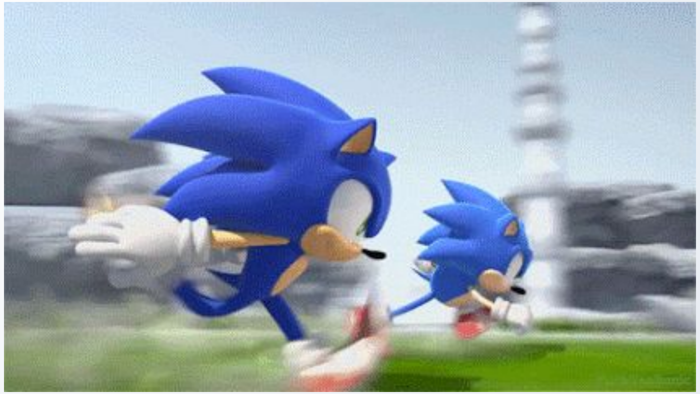
3. **Velocity Buffer Implementation**
    a. Velocity Buffer
    b. Post-processing Blur

1. What is Motion Blur?

- Capturing images is light entering camera over a **time interval**

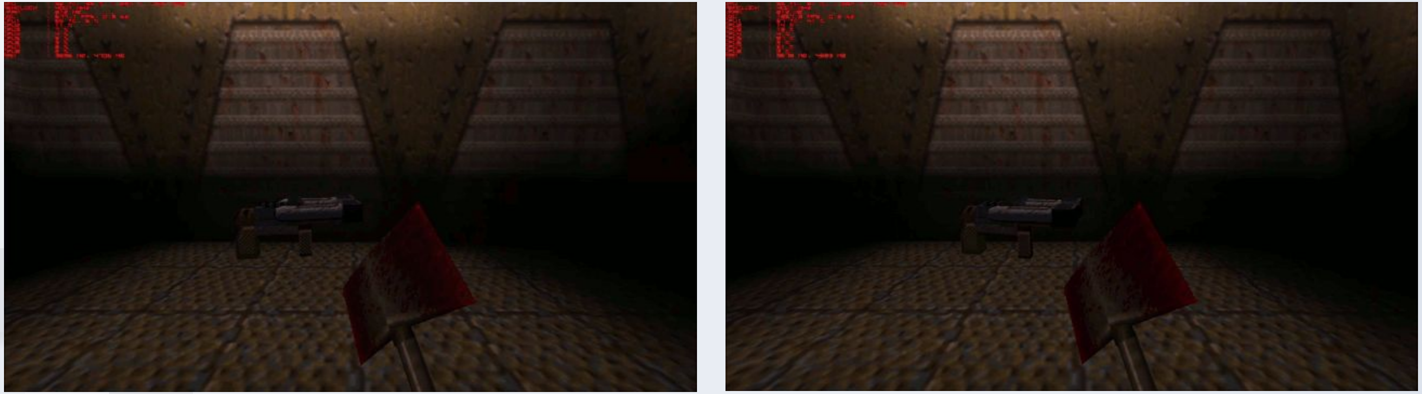
- **Not instantaneous,** so movement is recorded





Speed

Cinematic Effects



Smoothing

2. Implementation Methods

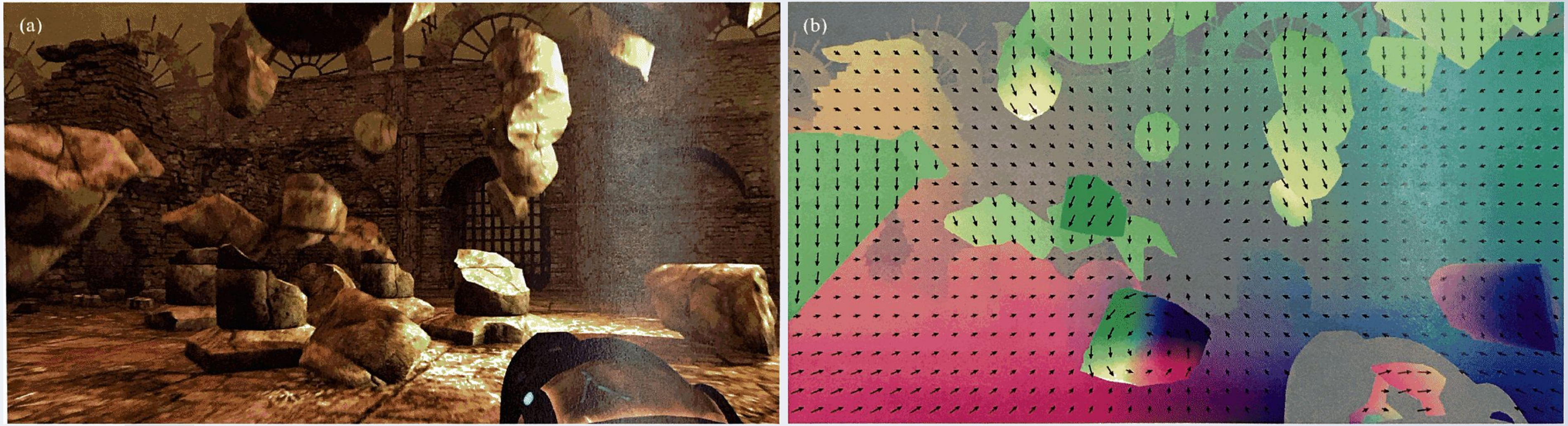
- Multiple ways to approach
    - Eg. Render many frames, average

- **Velocity Buffer Method**
    - Relatively inexpensive
    - Independent movement blurs properly

Velocity Buffer Method

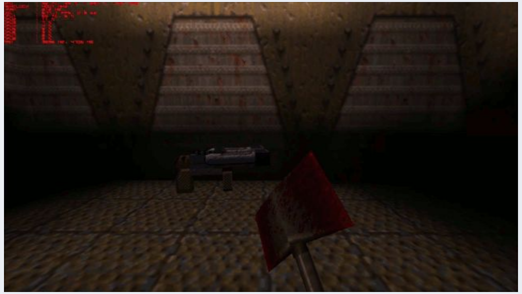
- Maintain a **Velocity Buffer:**
  - Holds a **velocity vector** for every pixel
  - Velocity vector represents how much that that surface will move between current/next frame

The Velocity Buffer

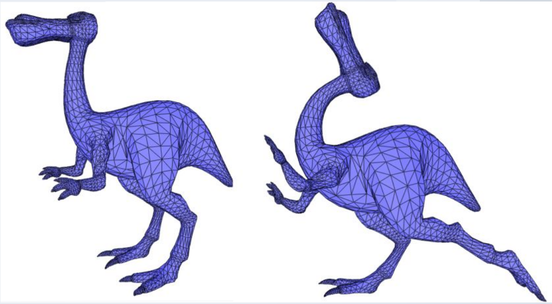


Sources of Movement

Moving camera



Object transform



Mesh deformation

But the Velocity Buffer handles all three :)

Getting Our Velocity Buffer Values

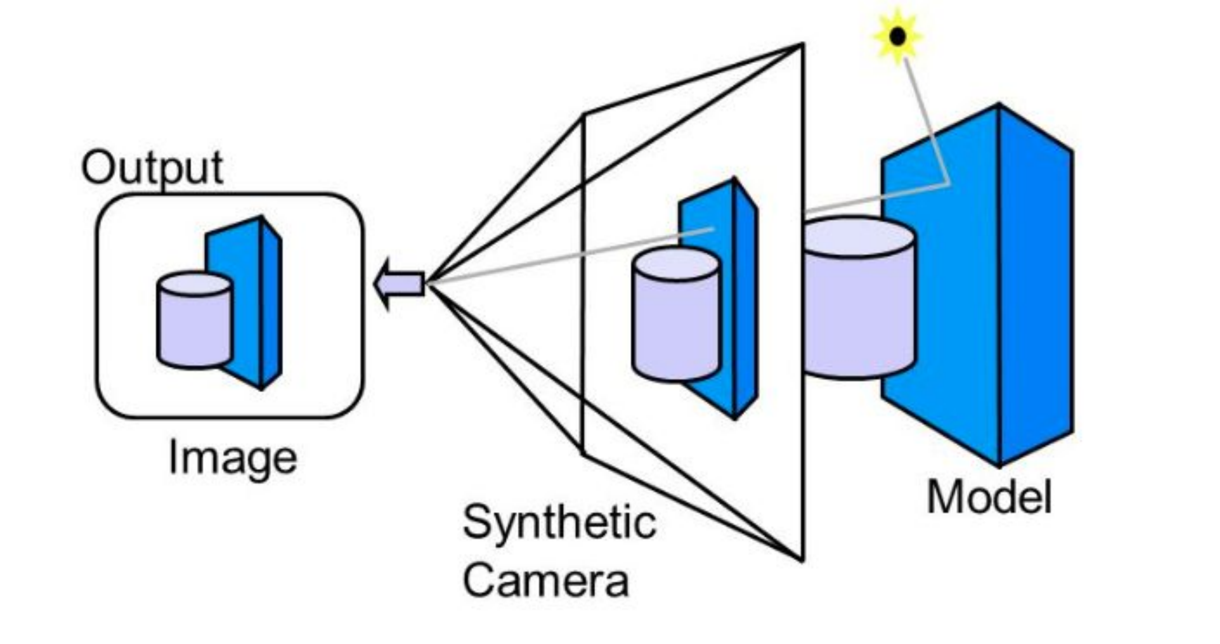
1. For each vertex
    a. Compute positions for current/previous frame
    b. Given positions, compute velocity
    c. Store velocity in buffer

Getting Our Velocity Buffer Values

$$\mathbf{p}_{\text{viewport}} = \mathbf{M}_{\text{viewport}} \mathbf{M}_{\text{projection}} \mathbf{M}_{\text{camera}}^{-1} \mathbf{M}_{\text{object}} \boldsymbol{p}_{\text{object}} .$$

- $\mathbf{P}_{\text{object}}$ - object space vertex position
- $\mathbf{M}_{\text{object}}$ - object space → world space
- $\mathbf{M}_{\text{camera}}$ - camera space → world space
- $\mathbf{M}_{\text{projection}}$ - projection matrix (perspective)
- $\mathbf{M}_{\text{viewport}}$ - scales to viewport dimensions
- $\mathbf{P}_{\text{viewport}}$ - viewport space vertex position

Getting Viewport Positions

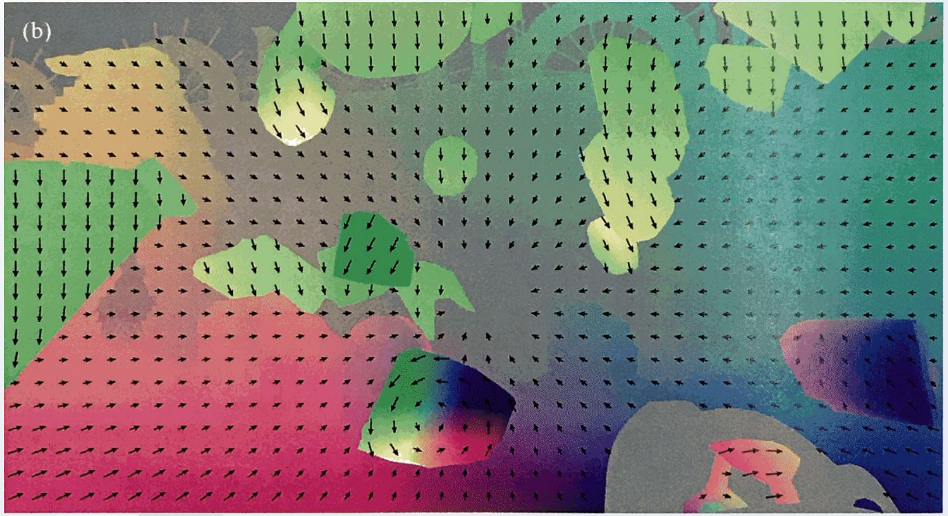


Computing Velocity

- Computing velocity: $v = \dfrac{d}{t}$

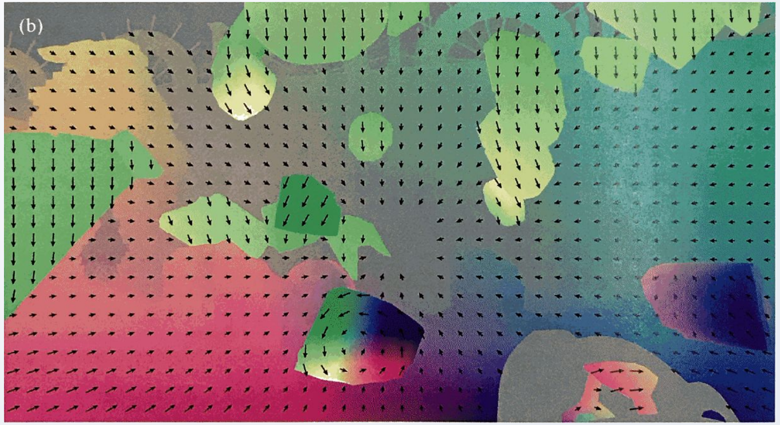
- The actual formula (p 361) is a little more complex
  - Scaling parameter
  - Normalized (time)
  - Clamped into [0, 1]

Storing Velocity



- Now **vertices** all have storable velocity values
- Velocities interpolated over triangles
- Every **pixel** has a storable velocity

Blur Postprocessing



How do we use this...

...for this?

Blur Postprocessing

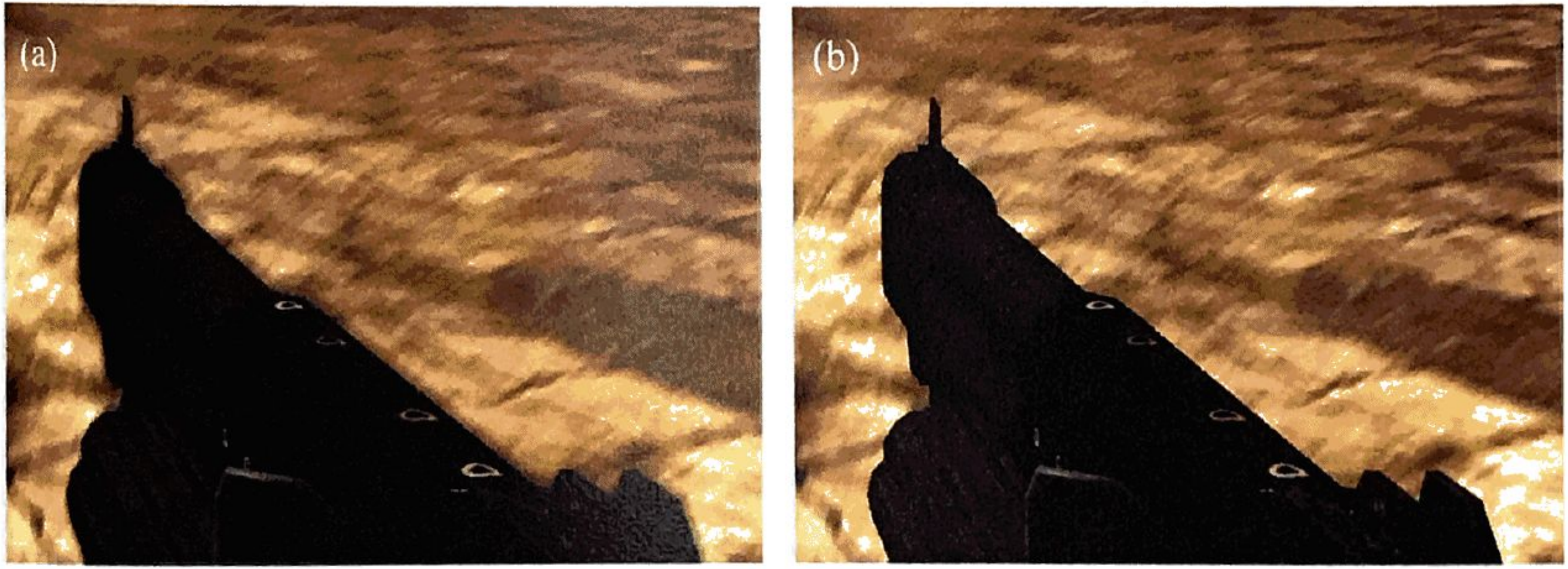
- For each pixel:

1. Reference pixel’s velocity vector
2. Read samples in +/- direction along vector
3. Average color information of samples

Blurring Sample Code

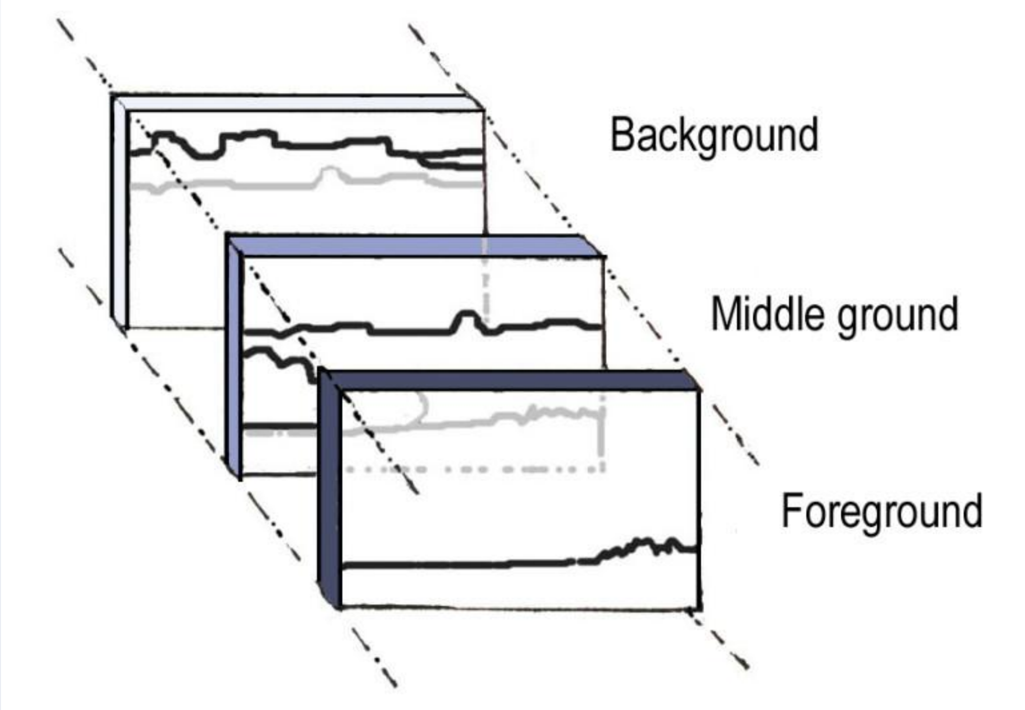
```
uniform TextureRect    colorBuffer;
uniform TextureRect    velocityBuffer;
uniform float          vstep;

float3 ApplySimpleMotionBlur(float2 pixelCoord)
{
   // Read color buffer and velocity buffer at center pixel.
   float3 color = texture(colorBuffer, pixelCoord).xyz;
   float2 velocity = texture(velocityBuffer, pixelCoord).xy * 2.0 - 1.0;

   // Add 8 more samples along velocity direction.
   for (int i = 1; i <= 4; i++)
   {
      float dp = float(i) * vstep;
      color += texture(colorTexture, pixelCoord + velocity * dp).xyz;
      color += texture(colorTexture, pixelCoord - velocity * dp).xyz;
   }

   // Return average of all samples.
   return (color * 0.1111111);
}
```

Implementation Extras (Read the Book)



DEPTH OF FIELD

Where’s the Focus



Turning our Motion Blur Method into DOF

- For each vertex
  - Compute distance from camera
  - If vertex is in **midground**, **no blur**
  - Else, **blur**
    - Sample around pixel
    - Average color information

Turning our Motion Blur Method into DOF

- For each vertex
  - Compute distance from camera
  - If vertex is in midground, no blur
  - Else, **blur**
    - Sample around pixel **with sample radius dependent on distance**
    - Average color information

Further Reading

[GPU Gems Chapter 23]

- Explains phenomenon in physics
- Overviews a couple implementations