
GameDevelopers Conference
MARCH 20-24
SAN JOSE, CALIFORNIA
WHAT'S NEXT
GDC:06
www.gdconf.com
GAME DEVELOPERS CHOICE AWARDS
INDEPENDENT GAMES FESTIVAL
GDC MOBILE
SERIOUS GAMES SUMMIT
GAME CONNECTION
CMP

DX10, Batching, and Performance Considerations

Bryan Dudash
NVIDIA Developer Technology

The Point of this talk

- *"The attempt to combine wisdom and power has only rarely been successful and then only for a short while. "- Albert Einstein*

- DX10 has many new features
  - Not just geometry shader!

- Opportunity to re-org your graphics architecture

Agenda

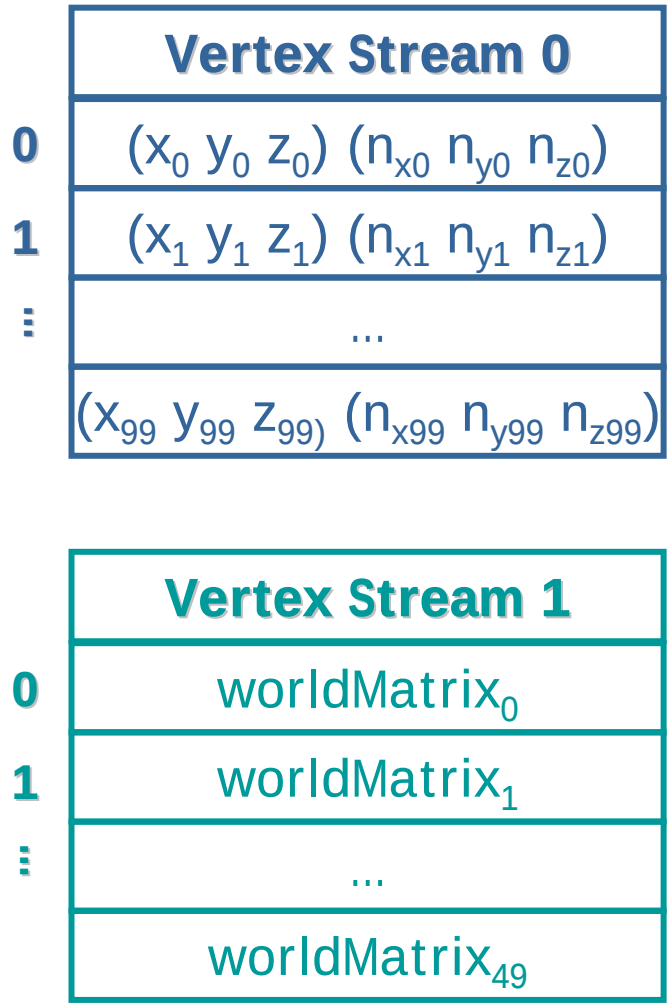
- Short History of DX9 performance
- DX10 Performance Potential
- Case Study: Geometry Particle System
- Case Study: Skinned Instanced Characters
- Conclusions

DX9 and Draw calls

- *“I wasted time, and now doth time waste me ”- William Shakespeare*
- Most DX9 games are CPU bound
- Often this is because of high #’s of draw calls
  - Developers have mostly learned this by now
- Often reducing draw calls isn’t trivial
  - Render state changes necessitate new draw

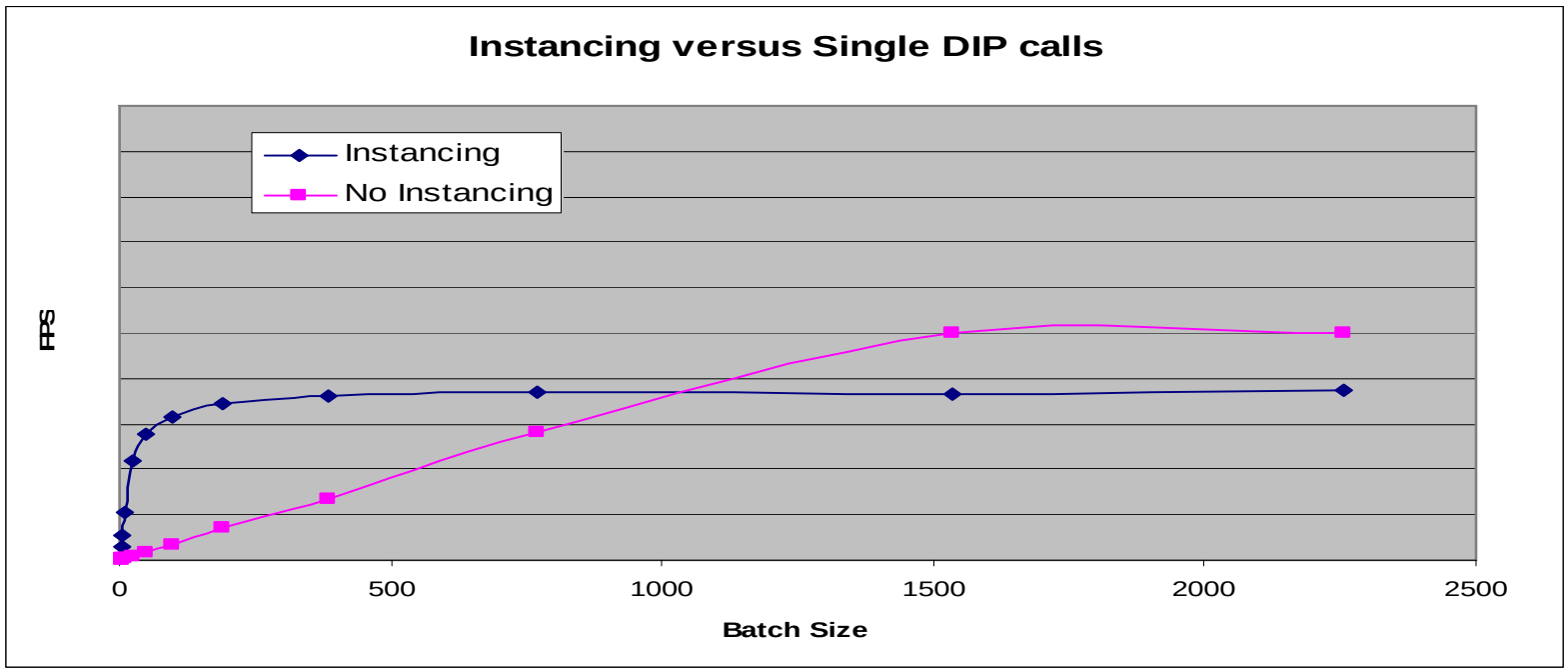
DX9 Instancing

- Not really in the API
  - ID3DDevice9::SetStreamSourceFreq()
- Set up 2 streams
- Modulus Vertex Stream 0
  - Base Mesh Stream
- Divide Vertex Stream 1
  - Instance Data Stream



DX9 Instancing Performance

- Test scene that draws 1 million diffuse shaded polys
- Changing the batch size, changes the # of drawn instances
- For small batch sizes, can provide an extreme win
- There is a fixed overhead from adding the extra data into the vertex stream
- The sweet spot changes based on CPU Speed, GPU speed, engine overhead, etc



……So what about Direct3D10?

How much faster is DX10?

"I was gratified to be able to answer promptly. I said I don't know. " – Mark Twain

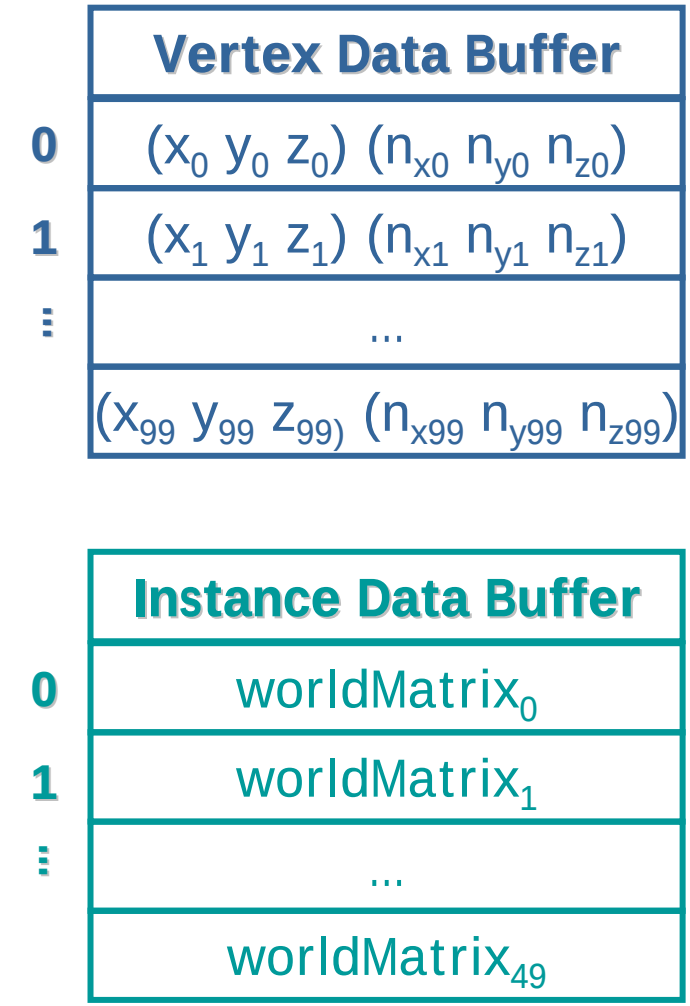
- But not just instancing
- Fundamentally alter graphics data flow
  - Increase parallelism
- Push more data processing to GPU

DX10 Performance Features

- General Instancing Support
- General data “buffer” concept
- Texture Arrays
- Geometry Shader
  - Yury will cover this in great detail later
- Stream Out

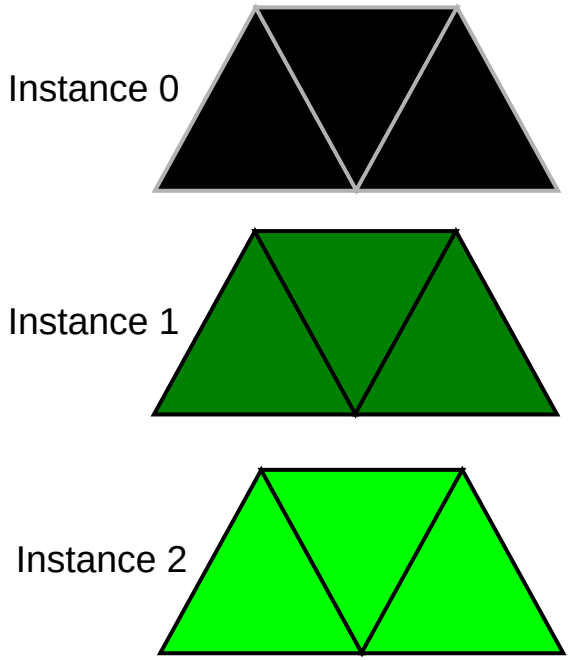
General Instancing Support

- Fundamentally unchanged
- But, fundamentally in the API
  - Single draw just a special case
- More useable due to other DX10 features



Instance ID

- Unique “system” value
- Incremented per instance
- Custom per instance processing

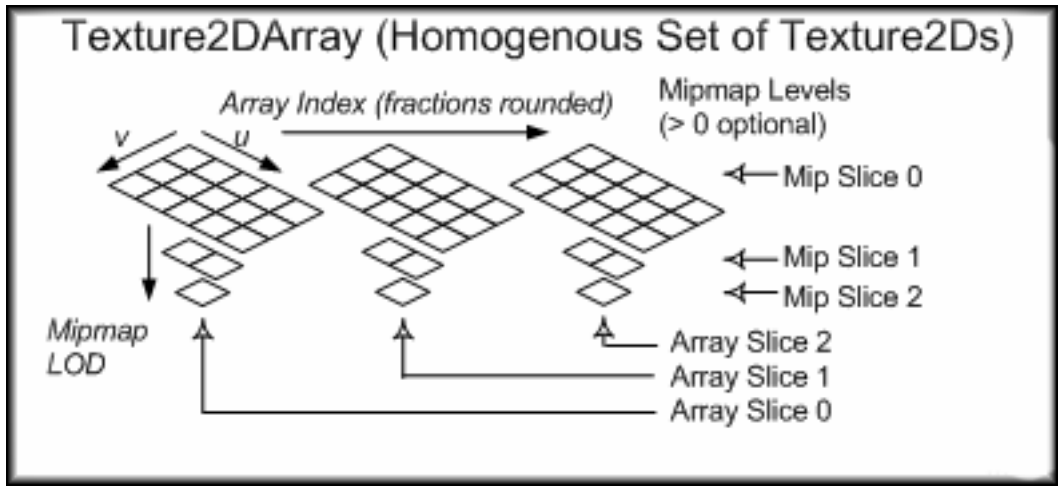


Color = float4(0,ID/2,0,0);

Data Buffer Object

- Input Assembler accepts
  - Vertex Buffer
  - Index Buffer
  - General Buffer
- Can only render to a general Buffer
  - And limited to 8k elements at a time
- Multiple passes can get you a R2VB

Texture Arrays

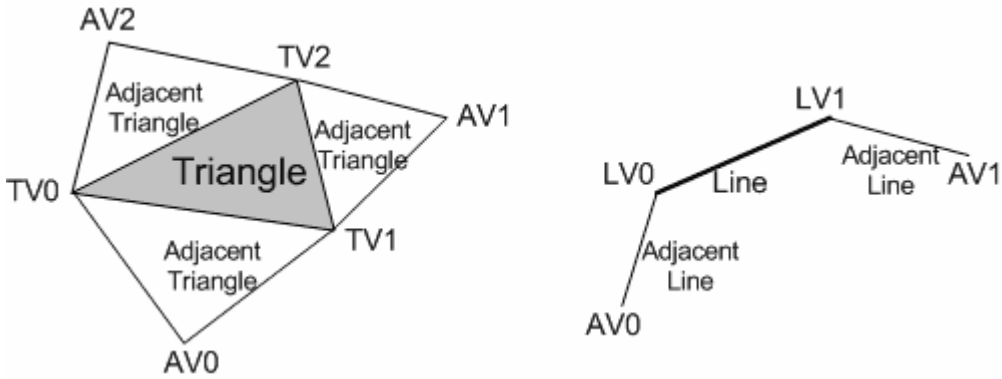


- All texture types can be used as an array
- Indexable from Shader
- Handy for instancing to store different maps for different instances

Texture Arrays and MRT

- Interesting tradeoff
- Texture Array is one big texture
  - With clamp constraints per “element” in the array
- Can output tris from GS to different slice
  - Possibly not writing to all slices
  - Adds extra VS/GS operations
- Regular MRT writes to all MRTs
  - Fixed B/W usage
  - But lower GS/VS ops

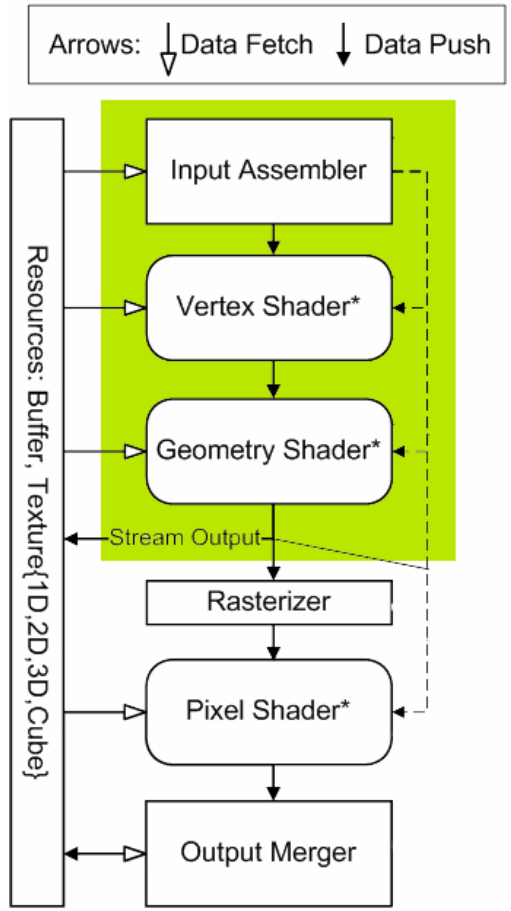
Geometry Shader



- Handy to allow us to offload MORE work from CPU
- Yury will go over GS potential in great depth

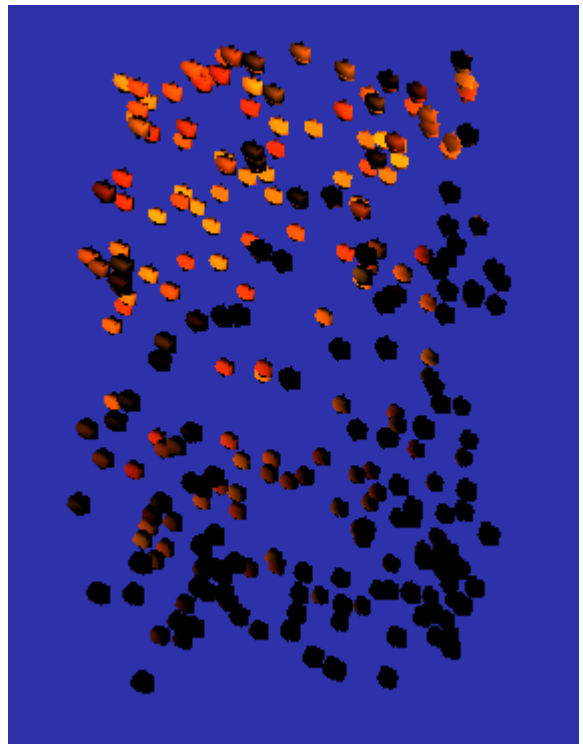
Stream Out

- Data output from Geometry Shader
  - Or Vertex Shader if GS is NULL
- Early out rendering pipeline before the Rasterization stage
- Allows us to fill dynamic vertex buffers and use in later pass.
  - Even as per instance data

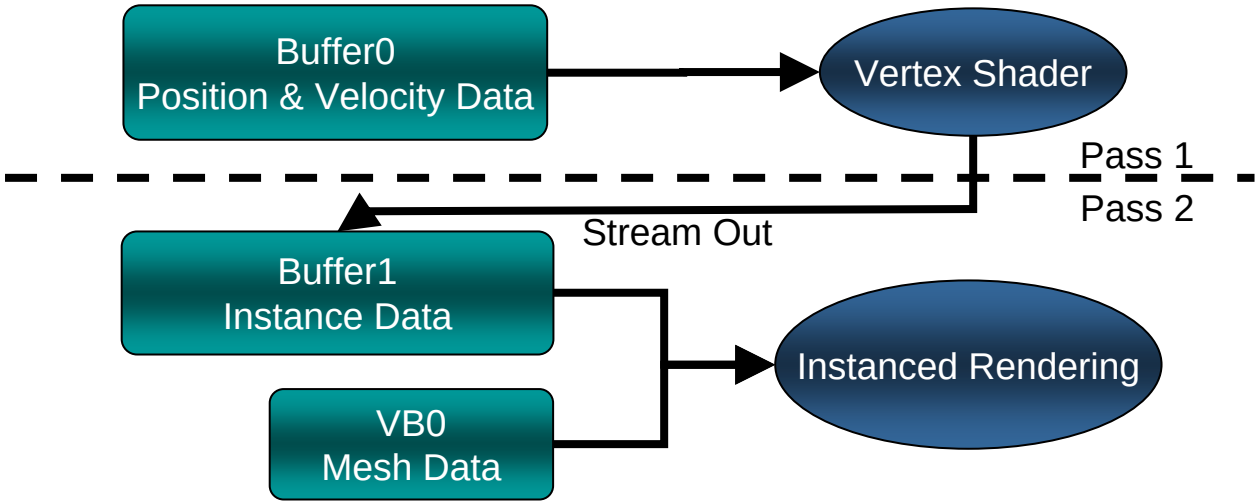


Case Study: Instanced Particles

- Particle simulation takes up a lot of CPU
- Updating a particle buffer costs bandwidth
- Often particle system just for effects
  - Game object don't need to know particle positions
- Geometry particles are cool!
  - More accurate lighting than sprites
  - Debris, broken glass, lava blobs



Basic Idea



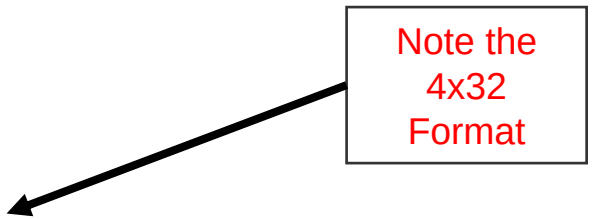
- Simulation done in first pass
- Position results used in second pass
  - Each particle is an instanced mesh
- Buffer0 and Buffer1 swapped every frame

Key Bits

- Stream Out
  - Stream out into an instance data buffer
  - Do particle simulation in VS
- Instance data
  - Vec4 – Position.xyz, lifetime
  - Vec3 – Velocity.xyz
- On CPU Maintain freelist
  - “inject” updates into instance stream
  - UpdateSubresource with a subrect

D3D10_INPUT_ELEMENT_DESC

```
{
    L"POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT,
    0, 0, D3D10_INPUT_PER_VERTEX_DATA, 0
},
{
    L"TEXTURE0", 0, DXGI_FORMAT_R32G32_FLOAT, 0,
    12, D3D10_INPUT_PER_VERTEX_DATA, 0
},
{
    L"NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT,
    0, 20, D3D10_INPUT_PER_VERTEX_DATA, 0
},
{
    L"particlePosition", 0, DXGI_FORMAT_R32G32B32A32_FLOAT,
    1, 0, D3D10_INPUT_PER_INSTANCE_DATA, 1
},
{
    L"particleVelocity", 0, DXGI_FORMAT_R32G32B32_FLOAT,
    1, 16, D3D10_INPUT_PER_INSTANCE_DATA, 1
},
```



Considerations

- Collision
  - Can handle simple collision primitives in shader
- Works for effects, not interactive objects
- Dead Particles
  - Assign a special NAN value to be interpreted as dead particle

Extensions

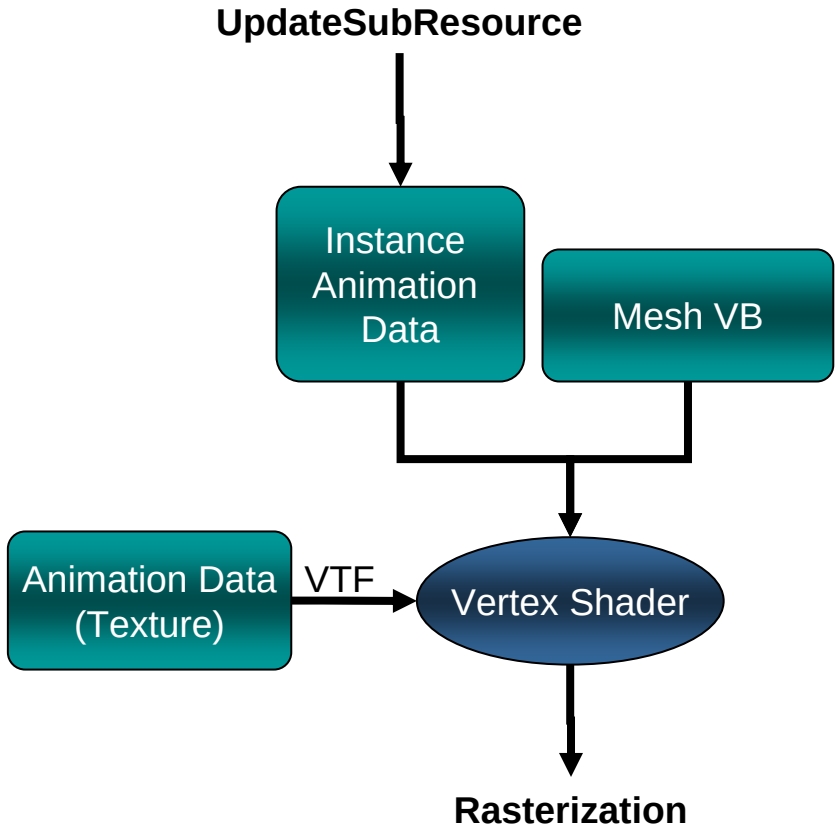
- Motion Blur
  - Setup final output to a RT
  - Use velocity data to calculate blur
    - Already have velocity from simulation
- Add in simple collision primitives
  - Sphere
  - Box
  - Terrain texture

Case Study: Skinned Instancing

- Would like to draw many animated characters
- Often these characters require upwards of a dozen draw calls EACH
- Lots of VS constants updated per draw
  - For palette skinning
- We'd like to batch together same mesh characters

Basic Idea

- Encode all animations into a texture
- A single character mesh
  - Contains same info for traditional palette skinning
- Each instance uses different animation
- Time controlled by CPU

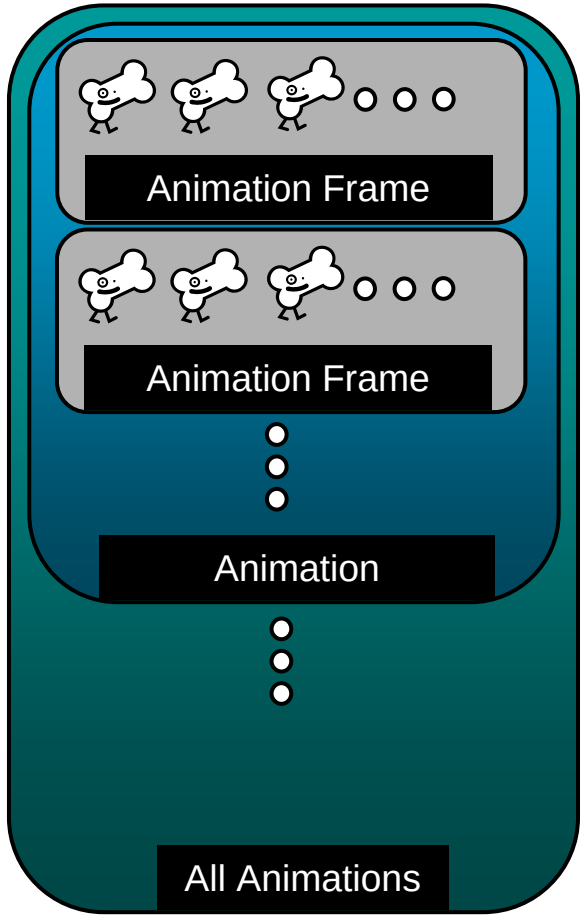


Key Bits

- Vertex Texture (VTF)
  - All animations
- Vertex Mesh Stream (static)
  - Vertex Data (ref pose)
  - Bone indices & weights
- Instance Stream (dynamic)
  - Animation offset
  - Frame offset
  - Time lerp

Animation Texture

- A “texel” is a row of the bone matrix
  - 4 texels form a single bone
- Example
  - 50 bone, 60 frame animation
    - 12,000 pixels
  - Easily stored in a 128x128



Animation Texture

- Cannot be 1D Texture or generic Buffer
  - Max size is 8192
- Could be a Texture Array
- Thus we encode our data linearly into a 2D texture

Load Bone HLSL Function

```
// Calculate a UV for the bone for this vertex
float2 uv = float2(0,0);

// if this texture were 1D, what would be the offset?
uint baseIndex = animationOffset + frameOffset + (4*bone);

// Now turn that into 2D coords
uint baseU = baseIndex%g_InstanceMatricesWidth;
uint baseV = baseIndex/g_InstanceMatricesWidth;
uv.x = (float)baseU / (float)g_InstanceMatricesWidth;
uv.y = (float)baseV / (float)g_InstanceMatricesHeight;

// Note that we assume the width of the texture is an even multiple of 4,
// otherwise we'd have to recalculate the V component PER lookup
float2 uvOffset = float2(1.0/(float)g_InstanceMatricesWidth,0);

float4 mat1 = g_txInstanceMatrices.Sample( g_samPoint,float4(uv.xy,0,0));
float4 mat2 = g_txInstanceMatrices.Sample( g_samPoint,float4(uv.xy + uvOffset.xy,0,0));
float4 mat3 = g_txInstanceMatrices.Sample( g_samPoint,float4(uv.xy + 2*uvOffset.xy,0,0));
float4 mat4 = g_txInstanceMatrices.Sample( g_samPoint,float4(uv.xy + 3*uvOffset.xy,0,0));

return float4x4(mat1,mat2,mat3,mat4);
```

Considerations

- This example is necessarily simple
  - Non-main characters/cutscenes
  - Real games have lots of data dependencies
  - Physics/Collision
- In game cutscenes?
- Processing and data loads onto GPU
  - But GPU is most often idle

Extensions

- Use Texture Array to store single animation in a slice
- Use TextureArray to encode multiple maps
  - Normals as well as Albedo
- Conditionally kill geometry in GS
  - Armor, Shields, etc
- Animation palette evaluation in GPU pass
  - Output the animation textures.

Conclusions

- *“Deliberation is the work of many men. Action, of one alone. “ – Charles De Gaulle*
- Instancing is more useful in DX10
- Working with data is easier
  - Think about how you can restructure your data
- More opportunity for GPU simulation

Questions?

- bdudash@nvidia.com