Game Engine Design

tum.3D

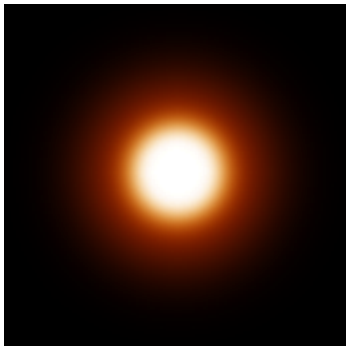
computer graphics & visualization

Assignment 9

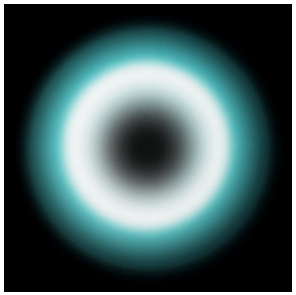
- This week: Sprites!
  (So we can shoot the enemies soon)

Sprites

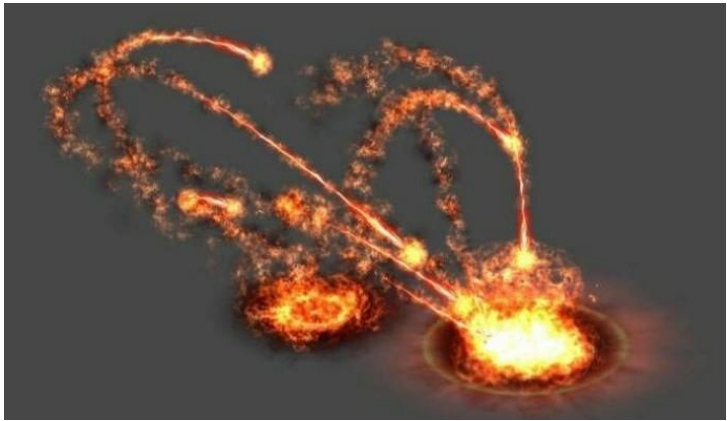
- How to render a „ball of plasma“ ?
  - Semi-transparent
  - Volumetric
  - → Mesh not appropriate!



- Simple hack: Replace by 2D image!
  - Looks the same from all directions anyway

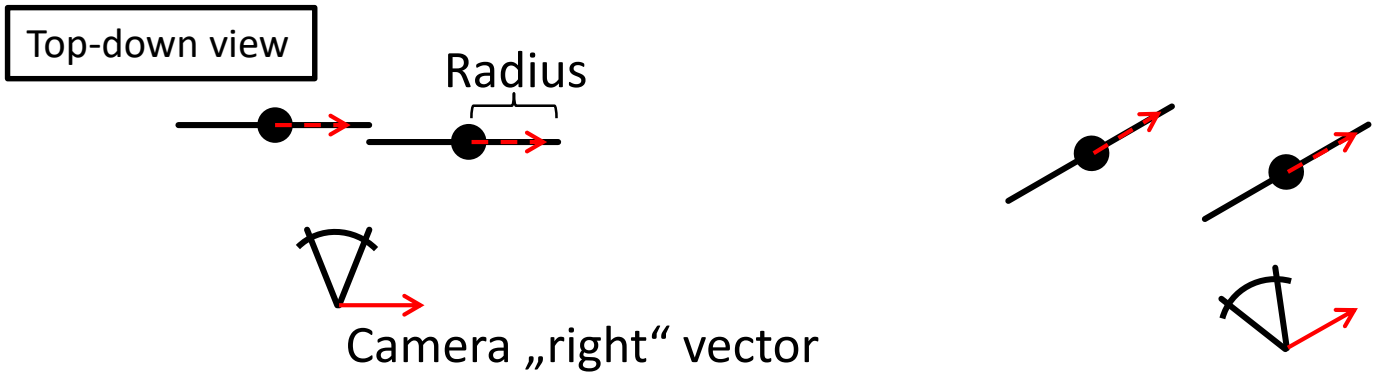


- Other typical use: Particle Systems
  - Smoke
  - Explosions



Sprite Rendering

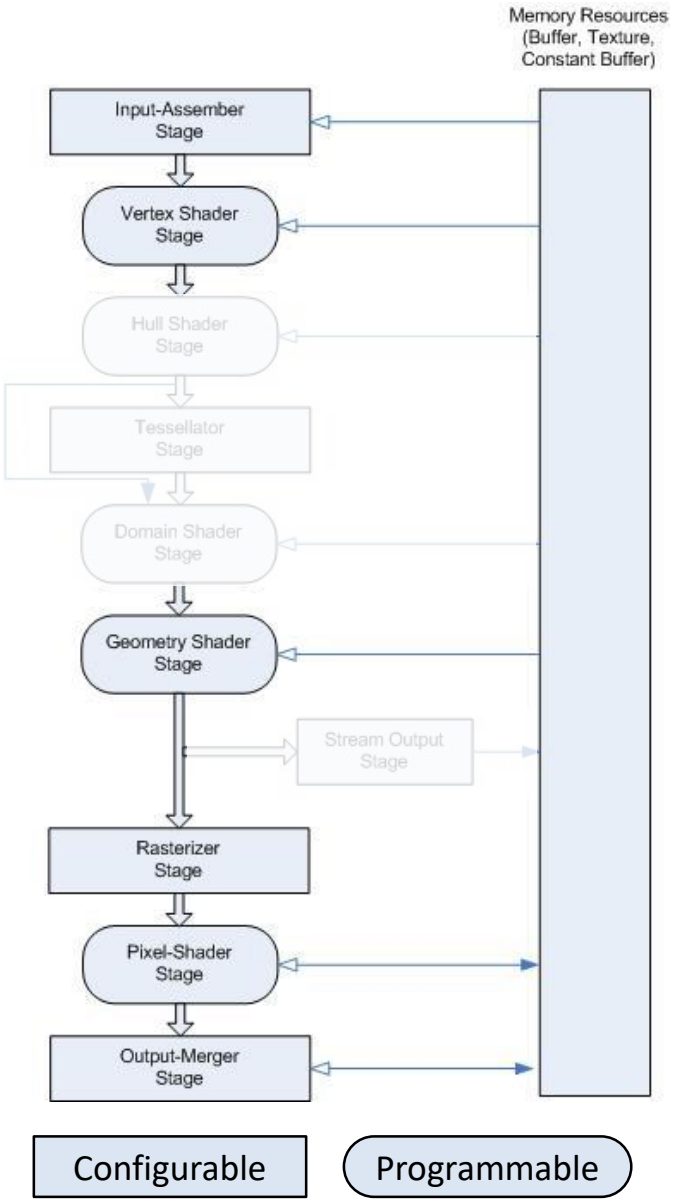
- Rendering: Camera-aligned quad



- Sprite definition:

```
struct SpriteVertex
{
    XMFLOAT3 Position;
    float Radius;
    int TextureIndex;
};
```
C++

Geometry Shader



- **Input Assembler**
- **Vertex Shader**
- **Geometry Shader**
  - optional
  - executed once **per primitive**
  - can **create primitives**
- **Rasterizer**
- **Pixel Shader**
- **Output Merger**

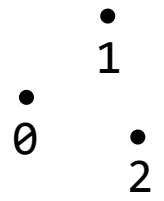
http://msdn.microsoft.com/en-us/library/windows/desktop/ff476882%28v=vs.85%29.aspx

What is a D3D Primitive?

- Primitive types and topologies:

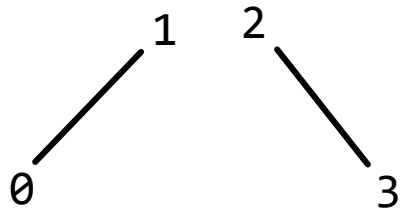
```
IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_*);
```

Points

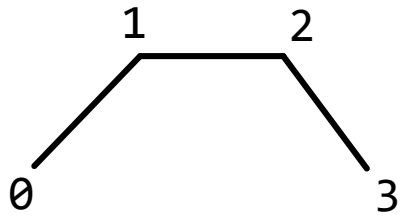


POINT_LIST

Lines

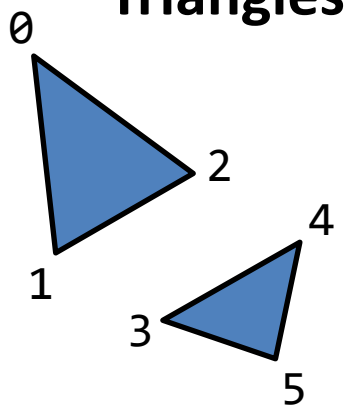


LINE_LIST

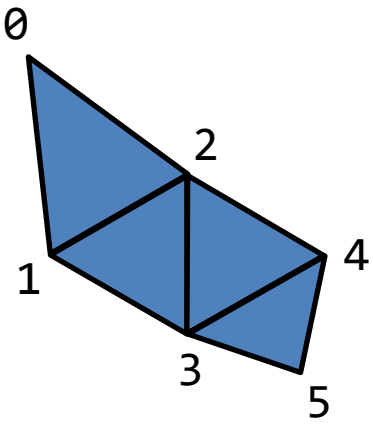


LINE_STRIP

Triangles



TRIANGLE_LIST

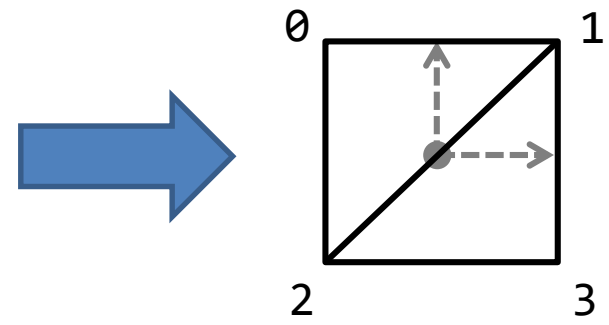


TRIANGLE_STRIP

http://msdn.microsoft.com/en-us/library/windows/desktop/bb205124%28v=vs.85%29.aspx

Geometry Shader: Point to Quad

Input: 1 Vertex
(point primitive)



Output: 4 Vertices
(triangle strip)

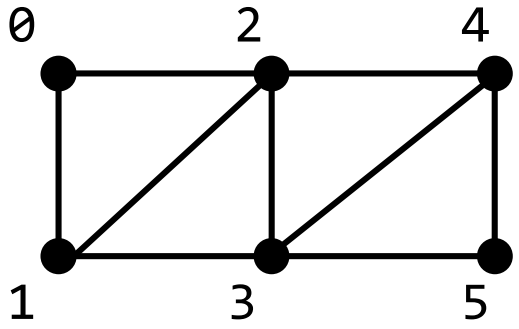
HLSL

```
struct SpriteVertex { float3 Position : POSITION; … };
struct PSVertex { float4 Position : SV_Position; … };

[maxvertexcount(4)]
void SpriteGS(point SpriteVertex vertex[1], inout TriangleStream<PSVertex> stream)
{
    …
}
```

- Maximum number of output vertices: 4
- Input: points (also possible: `line … [2]`, `triangle … [3]`)
- Output: triangle **strip** (also possible: `PointStream`, `LineStream`)
  Use `stream.Append(…)` to output a vertex

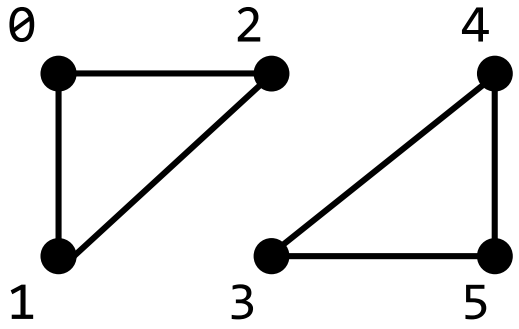
Geometry Shader Example

```
[maxvertexcount(6)]
void SpriteGS(point SpriteVertex vertex[1], inout TriangleStream<PSVertex> stream){
    PSVertex v;

    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output first vertex
    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output second vertex
    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output third vertex
    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output fourth vertex
    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output fifth vertex
    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output sixth vertex
}
```

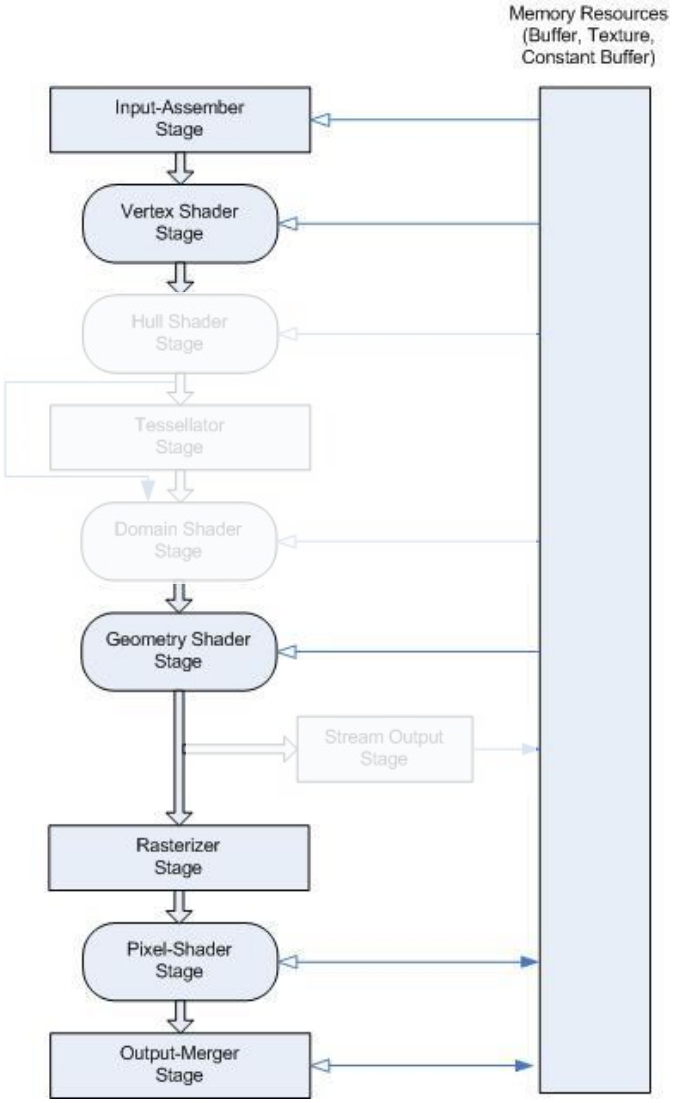


Geometry Shader Example

```
[maxvertexcount(6)]
void SpriteGS(point SpriteVertex vertex[1], inout TriangleStream<PSVertex> stream){
    PSVertex v;

    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output first vertex
    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output second vertex
    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output third vertex

    stream.RestartStrip();  // begin new triangle strip

    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output first vertex of second triangle
    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output second vertex of second triangle
    v.Position = float4(…); // set transformed position
    stream.Append(v);       // output third vertex of second triangle
}
```



Sprite Rendering – Overview



IA: create vertices

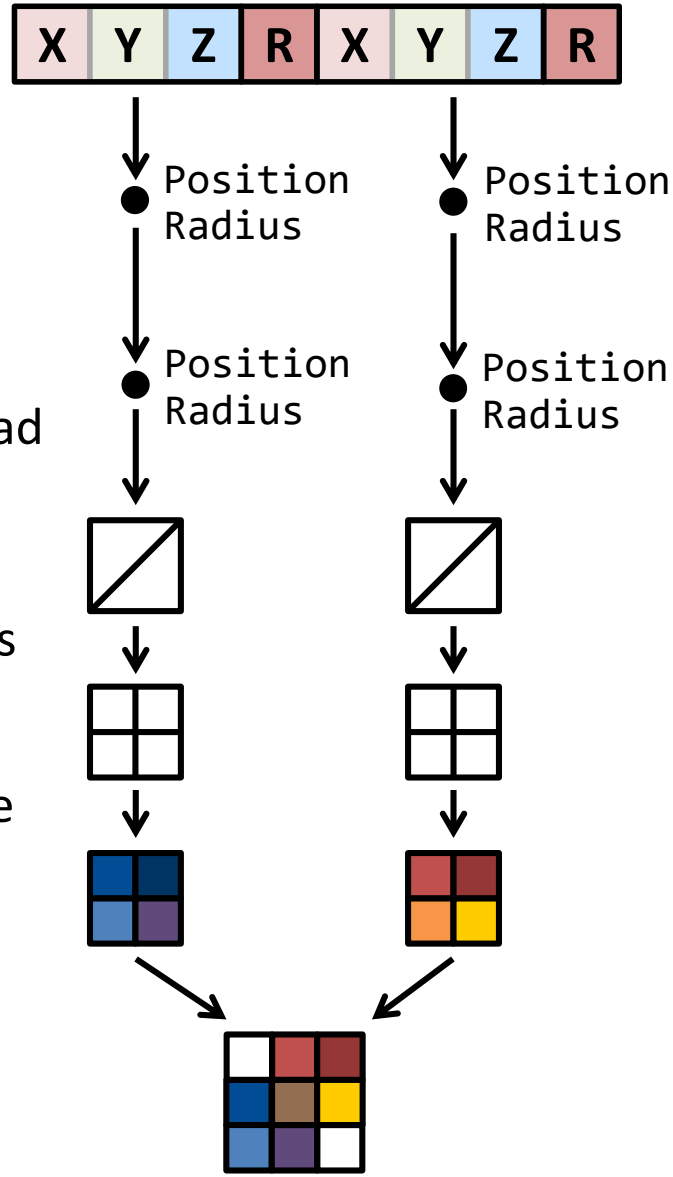
VS: pass through

GS: extrude to quad
transform

RS: generate pixels

PS: sample texture

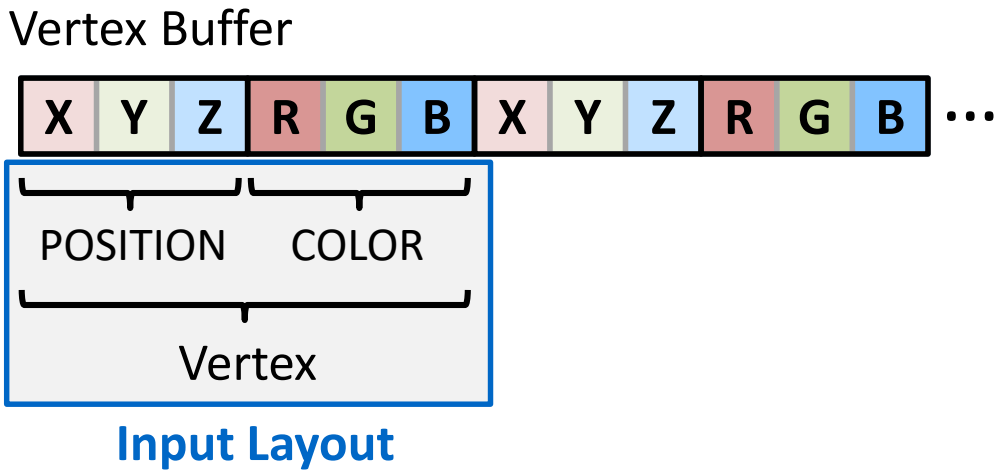
OM: blending



Reminder: Input Layout

- Defines the layout of data in a Vertex Buffer

```
struct MyVertex                    C++
{
    XMFLOAT3 Position;
    XMFLOAT3 Color;
};
```



- Also defines a name for each element, called **semantic**
  - Used to pass data to the Vertex Shader
- Input Layout in words:
  „one vertex is:
  a vector of 3 floats called `POSITION`,
  followed by a vector of 3 floats called `COLOR`“

Reminder: Creating an Input Layout

```
HRESULT ID3D11Device::CreateInputLayout(
    const D3D11_INPUT_ELEMENT_DESC *pInputElementDescs, // array of element
    UINT NumElements,                                   //   descriptions
    const void *pShaderBytecodeWithInputSignature,      // shader input
    SIZE_T BytecodeLength,                              //   signature
    ID3D11InputLayout **ppInputLayout                   // output
);
```

```
struct D3D11_INPUT_ELEMENT_DESC {
  LPCSTR                          SemanticName;      // element name (string)
  UINT                            SemanticIndex;     // usually 0
  DXGI_FORMAT                     Format;            // data type, DXGI_FORMAT_*
  UINT                            InputSlot;         // always 0 (for now)
  UINT                            AlignedByteOffset; // byte offset within vertex
  D3D11_INPUT_CLASSIFICATION InputSlotClass;    // D3D11_INPUT_PER_VERTEX_DATA
  UINT                            InstanceDataStepRate; // always 0 (for now)
};
```

Hint: Use `D3D11_APPEND_ALIGNED_ELEMENT` for `AlignedByteOffset`!

http://msdn.microsoft.com/en-us/library/windows/desktop/ff476512%28v=vs.85%29.aspx

Reminder: Creating an Input Layout

```
HRESULT ID3D11Device::CreateInputLayout(
    const D3D11_INPUT_ELEMENT_DESC *pInputElementDescs, // array of element
    UINT NumElements,                                   //   descriptions
    const void *pShaderBytecodeWithInputSignature,      // shader input
    SIZE_T BytecodeLength,                              //   signature
    ID3D11InputLayout **ppInputLayout                   // output
);
```

- Input Signature: What the Vertex Shader expects as input (data types and semantics)
- Used to verify that Input Layout and Vertex Shader match

```
D3DX11_PASS_DESC passDesc;
m_pEffect->GetTechniqueByName("MyTech")->GetPassByName("P0")->GetDesc(&passDesc);

passDesc.pIAInputSignature     → pShaderBytecodeWithInputSignature
passDesc.IAInputSignatureSize → BytecodeLength
```


http://msdn.microsoft.com/en-us/library/windows/desktop/ff476512%28v=vs.85%29.aspx

Reminder: Semantics

- Semantics are used to pass data between pipeline stages, e.g. from Input Assember to Vertex Shader

HLSL

```
struct MyVertex
{
    float3 Pos      : POSITION; // matched to input layout via semantics
    float3 Color    : COLOR;
};

void SimpleVS(MyVertex input, out float4 output : SV_Position)
{
    output = mul(float4(input.Pos, 1), g_WorldViewProjection);
}
```

- Also used between Vertex and Geometry/Pixel Shader!

```
void SimplePS(float4 pos : SV_Position, out float4 color : SV_Target) {…}
```

- SV_* („system value“) semantics are magic

Reminder: Effect Variables

HLSL

```
cbuffer cbPerFrame
{
    float3 g_CameraRight;
    …
}
```

C++

```
ID3DX11EffectVectorVariable* pCameraRightEV = // get handle
    pEffect->GetVariableByName(„g_CameraRight“)->AsVector();

pCameraRightEV->SetFloatVector(…);              // set value
```

- Other types: `Scalar`, `Matrix`, `ShaderResource`
- Look at the `SAFE_GET_*` macros in `game.cpp`!

Blending - Transparency

- Review the lecture slides! „Blend over“:

```
result.rgb = src.rgb * src.a + dest.rgb * (1 – src.a)
result.a   = src.a   * 1     + dest.a   * (1 – src.a)
```

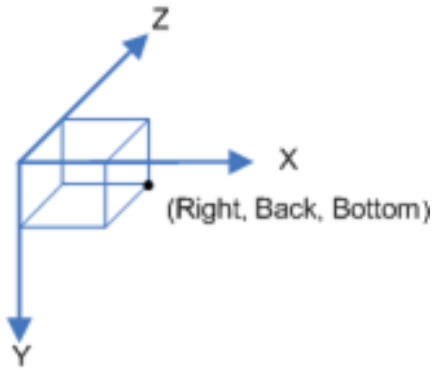
- **Sourc**e: The new pixel (Pixel Shader output)
- **Dest**ination: The previous framebuffer value
- Blending specification in `.fx`:

```
BlendState BSBlendOver
{
    BlendEnable[0]    = TRUE;
    SrcBlend[0]       = SRC_ALPHA;
    SrcBlendAlpha[0]  = ONE;
    DestBlend[0]      = INV_SRC_ALPHA;
    DestBlendAlpha[0] = INV_SRC_ALPHA;
};
```

Vertex Buffer Update

- Sprites usually move
  → We have to update the vertex buffer each frame!

```
void ID3D11DeviceContext::UpdateSubresource(
   ID3D11Resource *pDstResource, // resource (buffer/texture) to update
   UINT DstSubresource,          // always 0 for buffers
   const D3D11_BOX *pDstBox,     // region to update – see below
   const void *pSrcData,         // new data to write to the buffer
   UINT SrcRowPitch,             // irrelevant for buffers – set to 0
   UINT SrcDepthPitch            // irrelevant for buffers – set to 0
);

D3D11_BOX box;
box.left  = 0; box.right = vertexCount * sizeof(MyVertex);
box.top   = 0; box.bottom = 1;
box.front = 0; box.back   = 1;
```



Note: `[left, right)` is the buffer region to update (in bytes).
`top`, `bottom`, `front`, and `back` don't make sense for buffers (only 2D/3D textures), but still have to be specified as above!

http://msdn.microsoft.com/en-us/library/windows/desktop/ff476486%28v=vs.85%29.aspx

Questions?