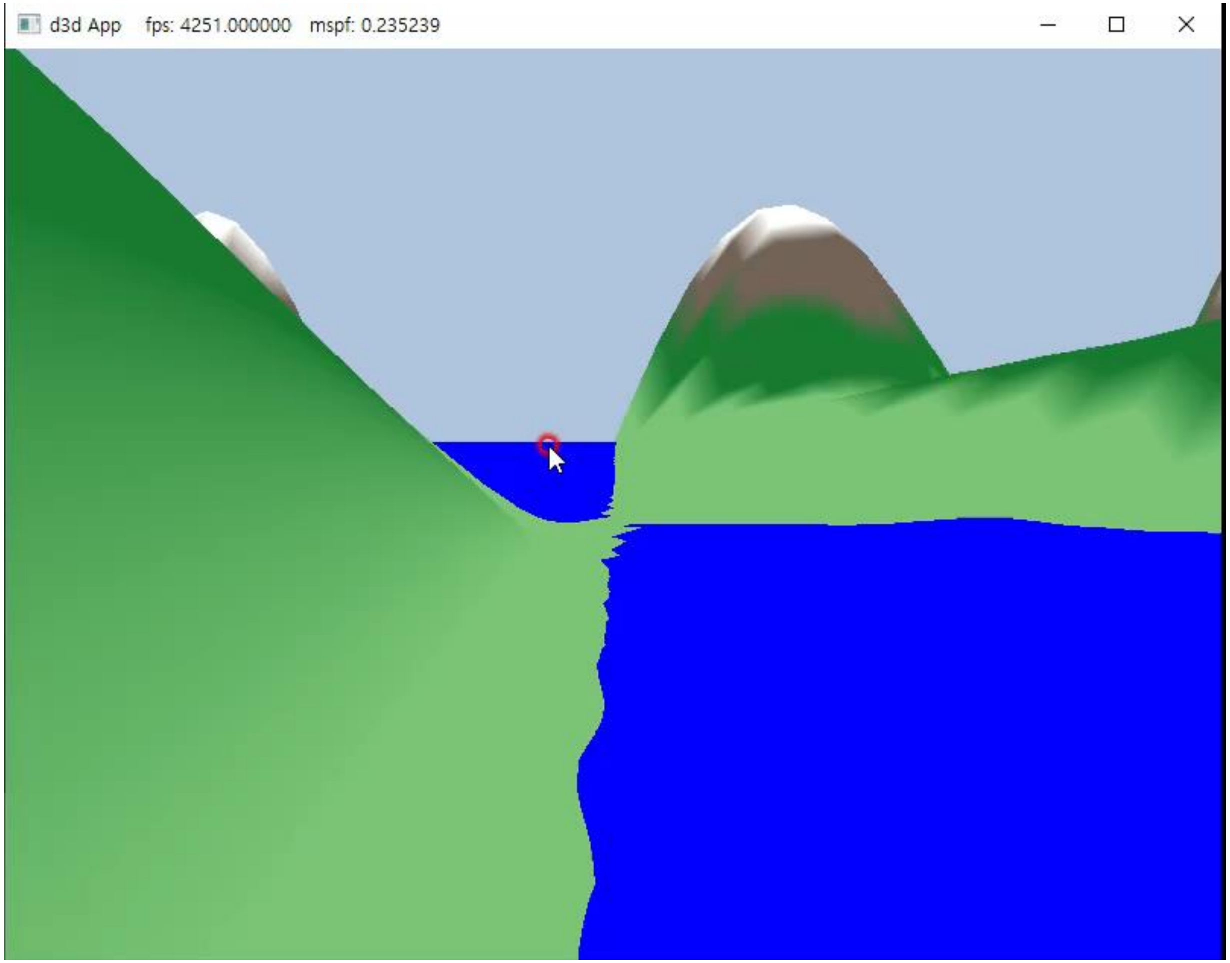
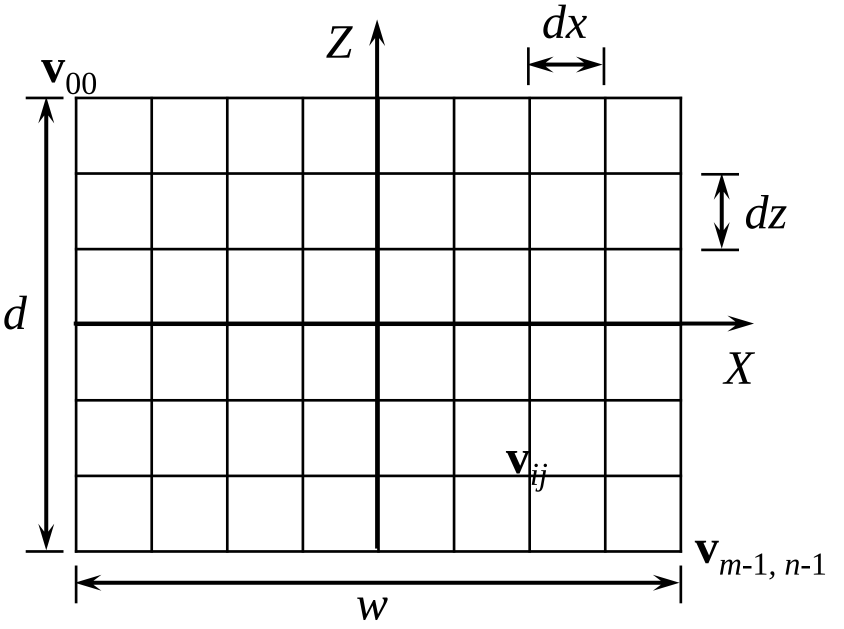
Demo

- Chapter 7 Drawing in Direct3D Part II\LandAndWaves



Generating the Grid Vertices

- The grid in the *xz*-plane.
  - A grid of $m \times n$ vertices induces (m-1)×(n-1) quads (2(m-1)×(n-1) triangles).



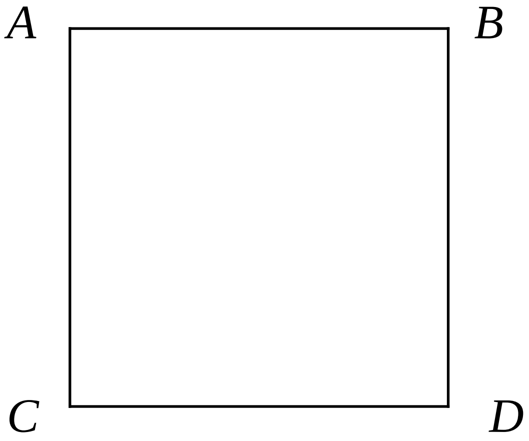
$$\mathbf{v}_{ij} = \left(-0.5w + jdx \quad 0.0 \quad 0.5d - idz\right)$$

Generating the Grid Indices

- Two triangles of the quad.

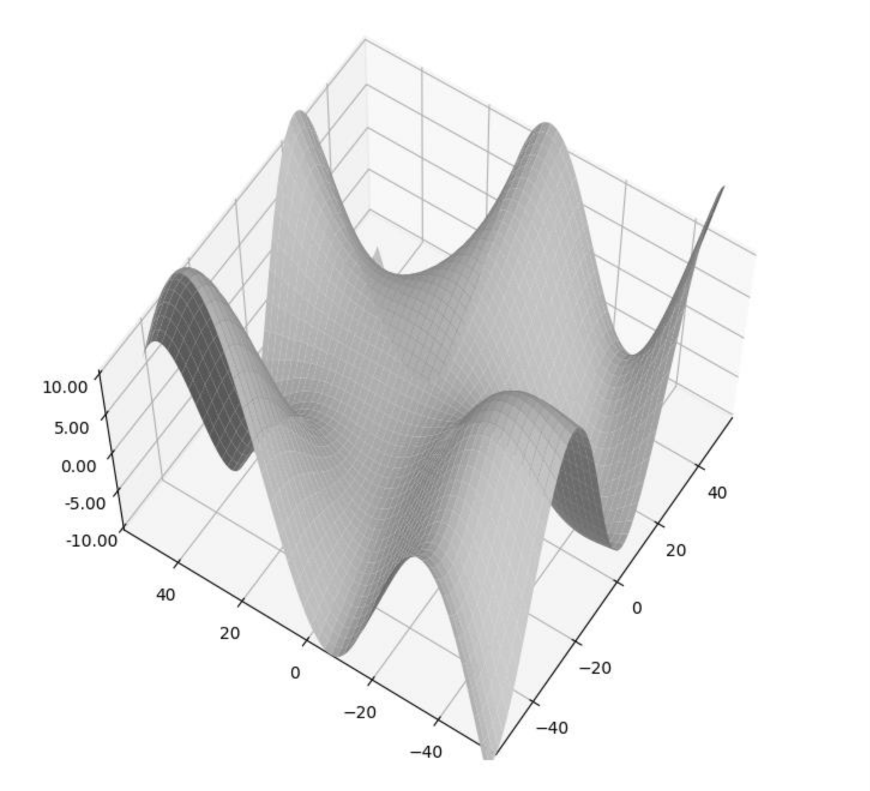
$$\Delta ABC = \left(i \cdot n + j \quad i \cdot n + j + 1 \quad (i+1)n + j \right)$$

$$\Delta CBD = \left((i+1)n + j \quad i \cdot n + j + 1 \quad (i+1)n + j + 1 \right)$$



Applying the Height Function

```
float LandAndWavesApp::GetHillsHeight(float x, float z)const {
    return 0.3f*(z*sinf(0.1f*x) + x*cosf(0.1f*z));
}
```



Generating the Land (1)

```
void LandAndWavesApp::BuildLandGeometry() {
  GeometryGenerator geoGen;
  GeometryGenerator::MeshData grid = geoGen.CreateGrid(160.0f, 160.0f, 50, 50);
// Extract the vertex elements we are interested and apply the height function
// to each vertex.  In addition, color the vertices based on their height so we
// have sandy looking beaches, grassy low hills, and snow mountain peaks.

   std::vector<Vertex> vertices(grid.Vertices.size());
   for(size_t i = 0; i < grid.Vertices.size(); ++i) {
       auto& p = grid.Vertices[i].Position;
       vertices[i].Pos = p;
       vertices[i].Pos.y = GetHillsHeight(p.x, p.z);
        // Color the vertex based on its height.
        if(vertices[i].Pos.y < -10.0f)     {           // Sandy beach color.
            vertices[i].Color = XMFLOAT4(1.0f, 0.96f, 0.62f, 1.0f);          }
        else if(vertices[i].Pos.y < 5.0f)  {           // Light yellow-green.
            vertices[i].Color = XMFLOAT4(0.48f, 0.77f, 0.46f, 1.0f);         }
        else if(vertices[i].Pos.y < 12.0f){           // Dark yellow-green.
            vertices[i].Color = XMFLOAT4(0.1f, 0.48f, 0.19f, 1.0f);          }
        else if(vertices[i].Pos.y < 20.0f){           // Dark brown.
            vertices[i].Color = XMFLOAT4(0.45f, 0.39f, 0.34f, 1.0f);         }
        else                               {           // White snow.
            vertices[i].Color = XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f);            }
   }
```

Generating the Land (2)

```
    const UINT vbByteSize = (UINT)vertices.size() * sizeof(Vertex);

    std::vector<std::uint16_t> indices = grid.GetIndices16();
    const UINT ibByteSize = (UINT)indices.size() * sizeof(std::uint16_t);

    auto geo = std::make_unique<MeshGeometry>();
    geo->Name = "landGeo";

    ThrowIfFailed(D3DCreateBlob(vbByteSize, &geo->VertexBufferCPU));
    CopyMemory(geo->VertexBufferCPU->GetBufferPointer(),
        vertices.data(), vbByteSize);

    ThrowIfFailed(D3DCreateBlob(ibByteSize, &geo->IndexBufferCPU));
    CopyMemory(geo->IndexBufferCPU->GetBufferPointer(),
        indices.data(), ibByteSize);

    geo->VertexBufferGPU = d3dUtil::CreateDefaultBuffer(md3dDevice.Get(),
        mCommandList.Get(), vertices.data(), vbByteSize,
        geo->VertexBufferUploader);

    geo->IndexBufferGPU = d3dUtil::CreateDefaultBuffer(md3dDevice.Get(),
        mCommandList.Get(), indices.data(), ibByteSize,
        geo->IndexBufferUploader);
```

Generating the Land (3)

```
    geo->VertexByteStride = sizeof(Vertex);
    geo->VertexBufferByteSize = vbByteSize;
    geo->IndexFormat = DXGI_FORMAT_R16_UINT;
    geo->IndexBufferByteSize = ibByteSize;

    SubmeshGeometry submesh;
    submesh.IndexCount = (UINT)indices.size();
    submesh.StartIndexLocation = 0;
    submesh.BaseVertexLocation = 0;

    geo->DrawArgs["grid"] = submesh;

    mGeometries["landGeo"] = std::move(geo);
}
```

Root CBVs (1)

- In this demo, we use root descriptors so that we can bind CBVs directly without having to use a descriptor heap.
    - The root signature needs to be changed to take two root CBVs instead of two descriptor tables.
    - No CBV heap is needed nor needs to be populated with descriptors.

Root CBVs (2)

```
void LandAndWavesApp::BuildRootSignature() {
    // Root parameter can be a table, root descriptor or root constants.
    CD3DX12_ROOT_PARAMETER slotRootParameter[2];

    // Create root CBV.
    slotRootParameter[0].InitAsConstantBufferView(0);
    slotRootParameter[1].InitAsConstantBufferView(1);

    // A root signature is an array of root parameters.
   CD3DX12_ROOT_SIGNATURE_DESC rootSigDesc(2, slotRootParameter, 0,
     nullptr,
     D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT);

    // Create a root signature with a single slot which points
    // to a descriptor range consisting of a single constant buffer
    ComPtr<ID3DBlob> serializedRootSig = nullptr;
    ComPtr<ID3DBlob> errorBlob = nullptr;
    HRESULT hr = D3D12SerializeRootSignature(&rootSigDesc,
       D3D_ROOT_SIGNATURE_VERSION_1,
       serializedRootSig.GetAddressOf(), errorBlob.GetAddressOf());
```

Root CBVs (3)

```
    if(errorBlob != nullptr) {
        ::OutputDebugStringA((char*)errorBlob->GetBufferPointer());
    }
    ThrowIfFailed(hr);

    ThrowIfFailed(md3dDevice->CreateRootSignature(
        0,
        serializedRootSig->GetBufferPointer(),
        serializedRootSig->GetBufferSize(),
        IID_PPV_ARGS(mRootSignature.GetAddressOf())));
}
```

Root CBVs (4)

- You can bind a CBV as an argument to a root descriptor using the following method:

```
void SetGraphicsRootConstantBufferView(
  [in] UINT                      RootParameterIndex,
  [in] D3D12_GPU_VIRTUAL_ADDRESS BufferLocation
);
```

Root CBVs (5)

```
void LandAndWavesApp::Draw(const GameTimer& gt) {
// …
// Bind per-pass constant buffer. We only need to do this once per-pass.
    auto passCB = mCurrFrameResource->PassCB->Resource();
    mCommandList->SetGraphicsRootConstantBufferView(1,
        passCB->GetGPUVirtualAddress());
    DrawRenderItems(mCommandList.Get(), mRitemLayer[(int)RenderLayer::Opaque]);
// …
}
```

Root CBVs (6)

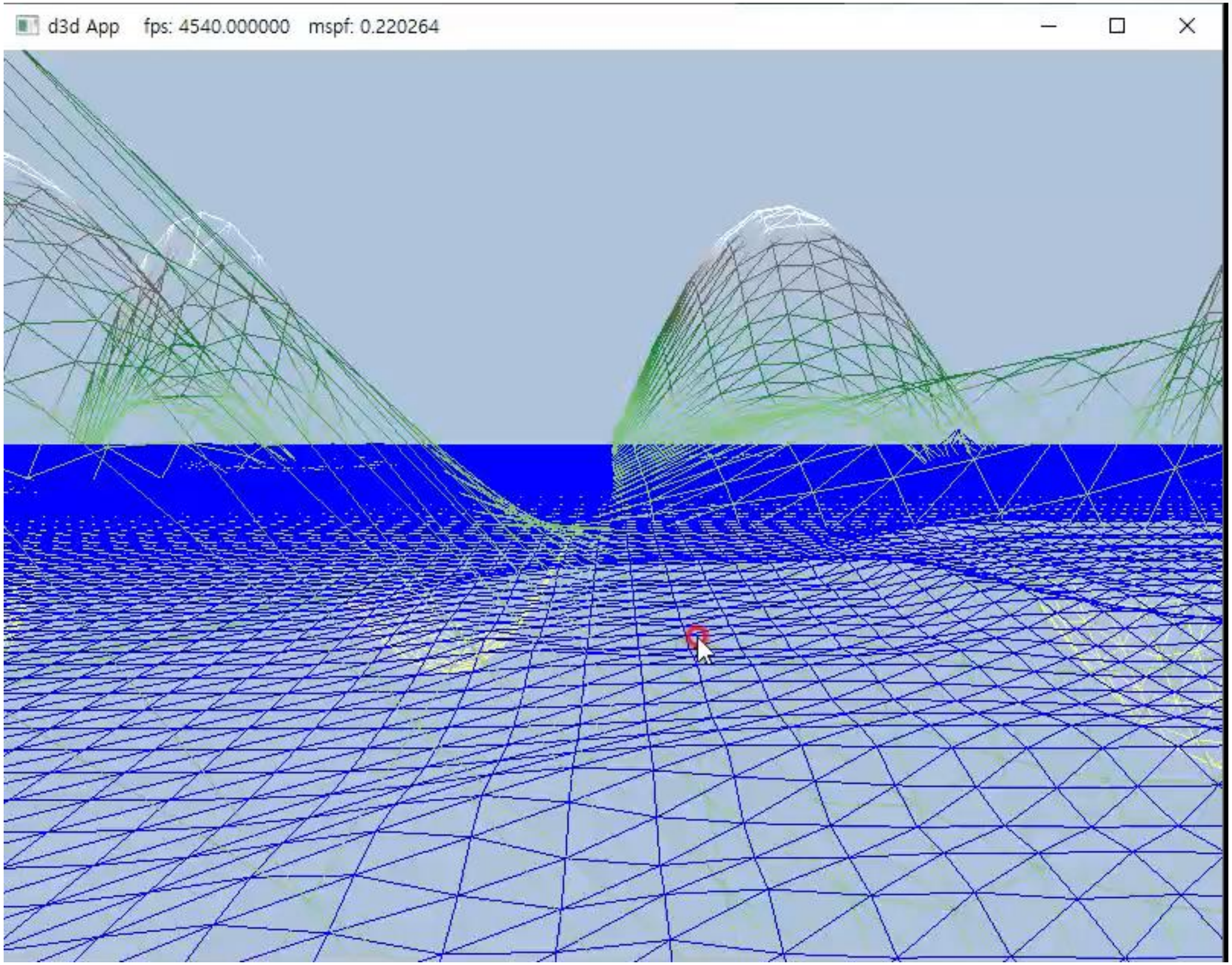
```
void LandAndWavesApp::DrawRenderItems(ID3D12GraphicsCommandList* cmdList,
    const std::vector<RenderItem*>& ritems) {
// …
    auto objectCB = mCurrFrameResource->ObjectCB->Resource();

    // For each render item...
    for(size_t i = 0; i < ritems.size(); ++i) {
        auto ri = ritems[i];
        // …

        D3D12_GPU_VIRTUAL_ADDRESS objCBAddress
            = objectCB->GetGPUVirtualAddress();
        objCBAddress += ri->ObjCBIndex*objCBByteSize;

        cmdList->SetGraphicsRootConstantBufferView(0, objCBAddress);
    // …
    }
// …
}
```

Dynamic Vertex Buffers (1)



Dynamic Vertex Buffers (2)

- Static vertex
    - We can use the default buffer resource.
- Dynamic vertex
    - Particle systems
    - We can use uploadable buffers, the user-defined **`UpladBuffer`** class.

```
void LandAndWavesApp::UpdateWaves(const GameTimer& gt) {
    // …
    // Update the wave simulation.
    mWaves->Update(gt.DeltaTime());
    // Update the wave vertex buffer with the new solution.
    auto currWavesVB = mCurrFrameResource->WavesVB.get();
    for(int i = 0; i < mWaves->VertexCount(); ++i) {
        Vertex v;
        v.Pos = mWaves->Position(i);
        v.Color = XMFLOAT4(DirectX::Colors::Blue);
        currWavesVB->CopyData(i, v);
    }
    // Set the dynamic VB of the wave renderitem to the current frame VB.
    mWavesRitem->Geo->VertexBufferGPU = currWavesVB->Resource();
}
```

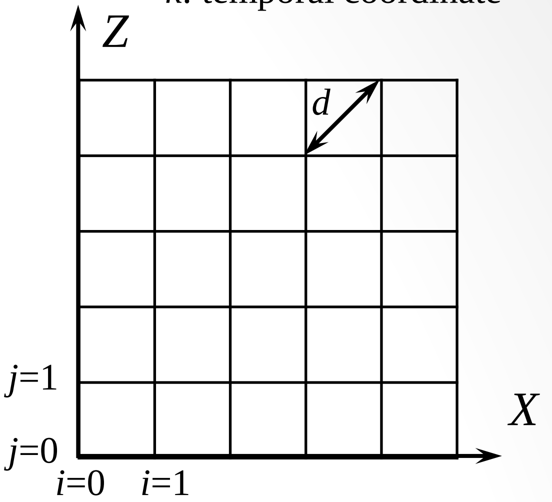
Dynamic Vertex Buffers (3)

- Fluid simulation
  - Wave equation

$$\frac{\partial^2 y}{\partial t^2} = c^2 \left(\frac{\partial^2 y}{\partial x^2} + \frac{\partial^2 y}{\partial z^2} \right) - \mu \frac{\partial y}{\partial t}$$

    - c: dimensions of distance per unit time (velocity)
    - μ: viscosity of the fluid (점도)

k: temporal coordinate



$$\frac{\partial y(i,j,k)}{\partial x} = \frac{\dfrac{y(i,j,k) - y(i-1,j,k)}{d} + \dfrac{y(i+1,j,k) - y(i,j,k)}{d}}{2} = \frac{y(i+1,j,k) - y(i-1,j,k)}{2d}$$

$$\frac{\partial y(i,j,k)}{\partial z} = \frac{y(i,j+1,k) - y(i,j-1,k)}{2d}$$

$$\frac{\partial y(i,j,k)}{\partial t} = \frac{y(i,j,k+1) - y(i,j,k-1)}{2t}$$

Dynamic Vertex Buffers (4)

$$\Delta\left(\frac{\partial y(i,j,k)}{\partial x}\right)=\frac{\dfrac{\partial y(i+1,j,k)}{\partial x}-\dfrac{\partial y(i-1,j,k)}{\partial x}}{2}=\frac{y(i+2,j,k)-2y(i,j,k)+y(i-2,j,k)}{4d}$$

$$\frac{\partial^2 y(i,j,k)}{\partial x^2}=\Delta\left(\frac{\partial y(i,j,k)}{\partial x}\right)\Big/\Delta x=\frac{y(i+2,j,k)-2y(i,j,k)+y(i-2,j,k)}{4d^2}$$

$$=\frac{y(i+1,j,k)-2y(i,j,k)+y(i-1,j,k)}{d^2}$$

$$\frac{\partial^2 y(i,j,k)}{\partial z^2}=\frac{y(i,j+1,k)-2y(i,j,k)+y(i,j-1,k)}{d^2}$$

$$\frac{\partial^2 y(i,j,k)}{\partial t^2}=\frac{y(i,j,k+1)-2y(i,j,k)+y(i,j,k-1)}{t^2}$$

Dynamic Vertex Buffers (5)

$$\frac{y(i,j,k+1)-2y(i,j,k)+y(i,j,k-1)}{t^2}$$

$$=c^2\frac{y(i+1,j,k)-2y(i,j,k)+y(i-1,j,k)}{d^2}+c^2\frac{y(i,j+1,k)-2y(i,j,k)+y(i,j-1,k)}{d^2}$$

$$-\mu\frac{y(i,j,k+1)-y(i,j,k-1)}{2t}$$

$$y(i,j,k+1)=\frac{4-8c^2t^2/d^2}{\mu t+2}y(i,j,k)+\frac{\mu t-2}{\mu t+2}y(i,j,k-1)$$

$$+\frac{2c^2t^2/d^2}{\mu t+2}\left[y(i+1,j,k)+y(i-1,j,k)+y(i,j+1,k)+y(i,j-1,k)\right]$$

Dynamic Vertex Buffers (6)

```
// Waves.h
class Waves {
// …
    void Update(float dt);
    void Disturb(int i, int j, float magnitude);
// …
    int mNumRows = 0;
    int mNumCols = 0;

    float mK1 = 0.0f;
    float mK2 = 0.0f;
    float mK3 = 0.0f;

    std::vector<DirectX::XMFLOAT3> mPrevSolution;
    std::vector<DirectX::XMFLOAT3> mCurrSolution;
    std::vector<DirectX::XMFLOAT3> mNormals;
    std::vector<DirectX::XMFLOAT3> mTangentX;
};
```

Dynamic Vertex Buffers (7)

```
// Waves.cpp
Waves::Waves(int m, int n, float dx, float dt, float speed, float damping) {
    mNumRows = m;
    mNumCols = n;

    float d = damping*dt + 2.0f;
    float e = (speed*speed)*(dt*dt) / (dx*dx);
    mK1 = (damping*dt - 2.0f) / d;
    mK2 = (4.0f - 8.0f*e) / d;
    mK3 = (2.0f*e) / d;
// …
}
void Waves::Disturb(int i, int j, float magnitude) {
// …
    float halfMag = 0.5f*magnitude;

    mCurrSolution[i*mNumCols+j].y     += magnitude;
    mCurrSolution[i*mNumCols+j+1].y   += halfMag;
    mCurrSolution[i*mNumCols+j-1].y   += halfMag;
    mCurrSolution[(i+1)*mNumCols+j].y += halfMag;
    mCurrSolution[(i-1)*mNumCols+j].y += halfMag;
}
```

Dynamic Vertex Buffers (8)

```
// Waves.cpp
void Waves::Update(float dt) {
    static float t = 0;
    t += dt;
    if( t >= mTimeStep ) {
        for(int i = 1; i < mNumRows-1; ++i) {
            for(int j = 1; j < mNumCols-1; ++j) {
                mPrevSolution[i*mNumCols+j].y =
                    mK1 * mPrevSolution[i*mNumCols+j].y +
                    mK2 * mCurrSolution[i*mNumCols+j].y +
                    mK3 * (mCurrSolution[(i+1)*mNumCols+j].y +
                          mCurrSolution[(i-1)*mNumCols+j].y +
                          mCurrSolution[i*mNumCols+j+1].y +
                          mCurrSolution[i*mNumCols+j-1].y);
            }
        }
    }
    std::swap(mPrevSolution, mCurrSolution);
    t = 0.0f;
// …
}
```