PipesGS Sample

Collapse All

This sample implements two vine generation algorithms on the GPU using the StreamOut capabilities of the Geometry Shader and DrawAuto.

Path

Source	*SDK root* \Samples\C++\Direct3D10\PipesGS
Executable	*SDK root* \Samples\C++\Direct3D10\Bin\ *x86 or x64* \PipesGS.exe

How the Sample Works

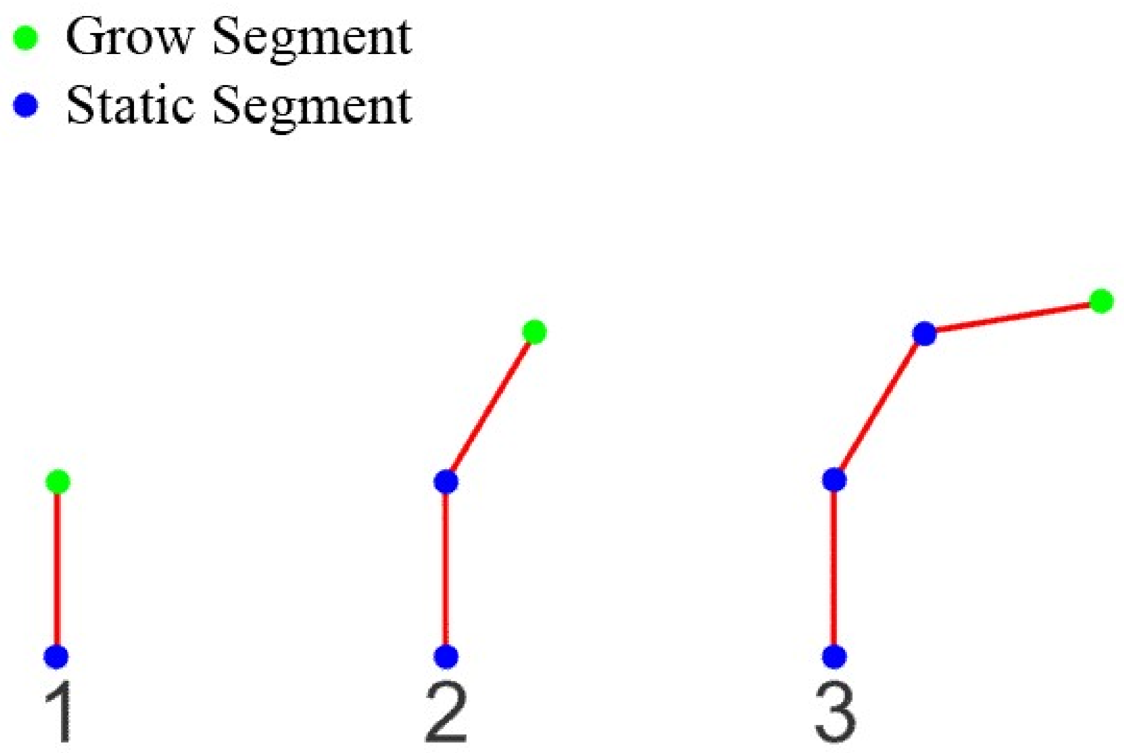
The vines for VinesGS are generated using two algorithms. The first algorithm starts the vines at the face centers of random mesh faces and grows the vines away from the normal of the face. The second starts the vines at the face centers of random mesh faces, but then grows the vines along the tangent of the face in the direction of one of the nearest face centers. The details of the algorithms can be found by looking through the geometry shader code for both types of vine generation.

The focus of the sample is geometry amplification and deamplification using the Geometry Shader. As with ParticlesGS, the geometry is handled in two passes. The first pass grows the pipes, while the second renders them.

Growing Pipes

For more details on geometry amplification in the Geometry Shader, see ParticlesGS.

For this sample, the pipes segments are tracked by a line list. Each point in the line list represents one pipe segment. The end of the pipe segment is the only area where a new pipe segment can grow from. This is labeled as a Grow point. All other points are either labeled as Start points (the start of a new vine) or Static points. Start points differentiate where one pipe stops and another pipe begins. Static points can only age and die when their timer is up. When the Geometry Shader encounters a Grow point, it coverts it to a Static point. It then outputs another Grow point a short distance away from the current point in a direction defined by the generation algorithm. For Static points, the shader keeps track of the time it's been alive as well as the total time that the vine it belongs to has been alive. When a Static point's timer runs out, it is not output to the stream-out buffer. This effectively kills the point.

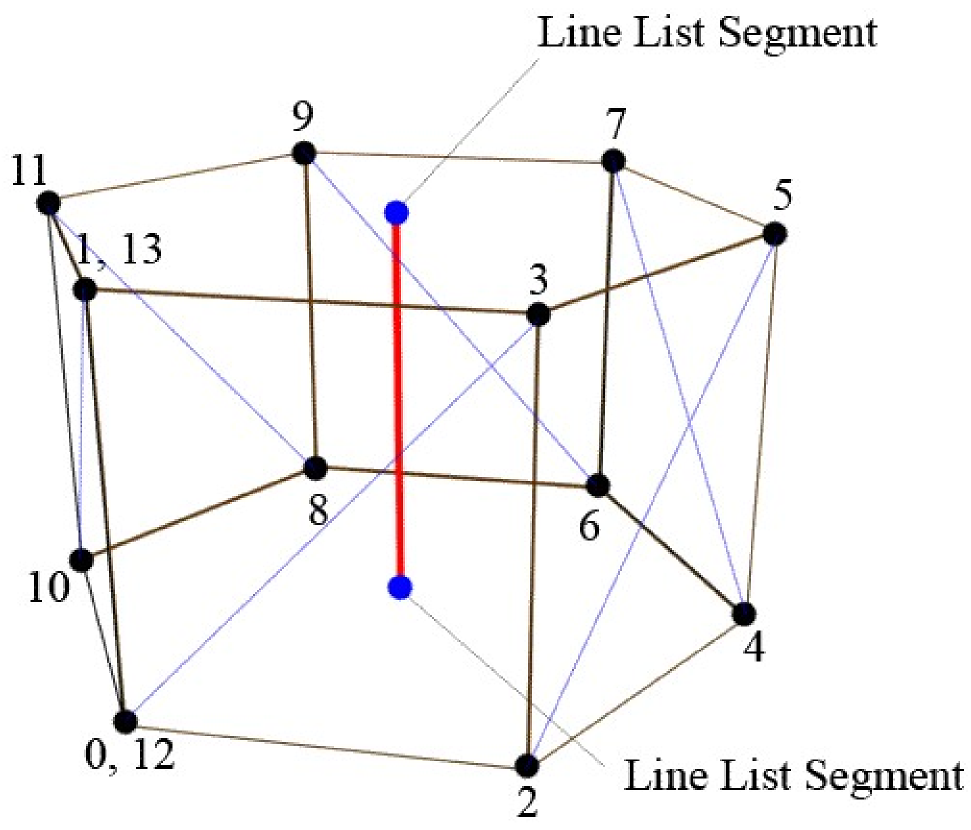


Growing Leaves

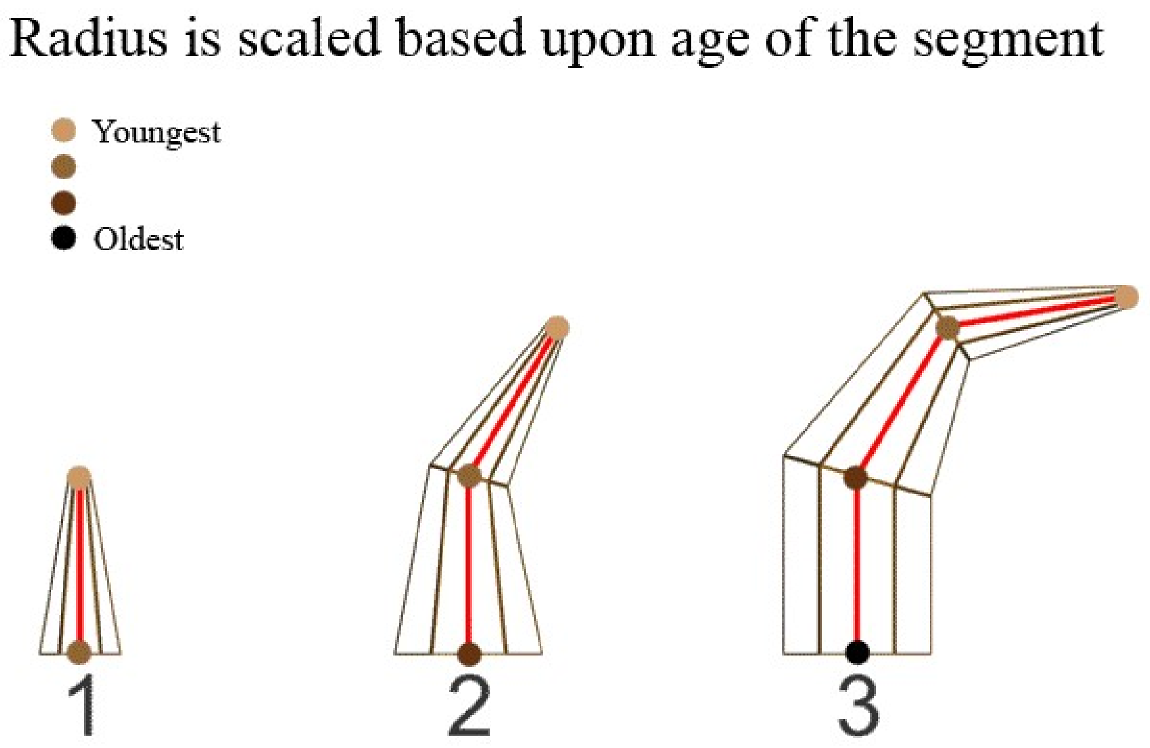
PipesGS also grows leaves along the length of the pipe. Leaves are handled with a separate variable in the point structure calls *leaves*. For each point that is added the line list, a random number in the range of [0..1] is selected inside the shader. If the number is below the leaf generation rate as defined by the application, the *Leaves* member of that point structure is set to a random number. Otherwise, *Leaves* is 0 for this point. During the rendering pass, the Geometry Shader renders a leaf with a random texture at that point if *Leaves* is non-zero.

Drawing the Pipes and Leaves

This particle system is composed of 5 different particle types with varying properties. Each particle type has it's own velocity and behavior and may or may not emit other particles.



Additionally points at each end of the cylinder are scaled according to the timing information of the closest point. Segments will go through the following life cycle. Young segments will grow until they reach their maximum radius. They will continue to age until they reach a time where they start to whither. During withering, the radius will be scaled down to zero. Once the radius is scaled to zero, the segment may die.



For each pair of points, the shader checks to see whether the first point's *Leaves* field is non-zero. If it is, a leaf is draw at this point. The leaf is made from a two-triangle strip that represents a quad oriented with the pipe at that point. The Z texture coordinate for the leaf quad is selected from the range of [1..5] based upon the value of *Leaves*. In the pixel shader, any leaves will fetch a texel from the indices [1..5] of the texture array, which have been preloaded as leaf textures. All pipe sections will use index 0 of the texture array, which is a bark texture.

Knowing How Many Pipe Segments are in the Buffer

Geometry Shaders can emit a variable amount of data each frame. Because of this, the sample has no way of knowing how many pipe segments are in the buffer at any given time. Using standard Draw calls, the sample would have to guess at the number of pipe segments to tell the GPU to draw. Fortunately, DrawAuto is designed to handle this situation. DrawAuto allows the dynamic amount of data written to streamout buffer to be used as the input amount of data for the draw call. Because this happens on the GPU, the CPU can advance and draw the pipes with no knowledge of how much geometry is actually in any of the buffers.