Motivation

Realistic rendering of water bodies such as rivers and oceans has proven to be one of the most difficult challenges in computer graphics. This poster focuses on simulating flowing water in complex river networks with various kinds of junctions.

Previous work includes several different kinds of particle systems [Yu et al. 2009] [Kipfer and Westermann 2006]. Particle systems are an intuitive approach for simulating water flow but often require large amounts of memory and computational power. We present an efficient technique for real-time rendering of complex river networks without using any kind of particle system. Instead, Bézier curves and streaming normal maps are used to simulate the flow of water through rivers.

Implementation

This technique has been implemented as an extension to the open source osgOcean nodekit [osgOcean], which is part of OpenSceneGraph.

Bézier point projection

Projecting a point \( T \) onto a Bézier curve \( B(t) \) is generally a complex problem to solve. However, if we only use quadratic order Bézier curves, the problem can be reduced to solving a cubic polynomial.

\[
\begin{align*}
\left( \frac{1}{2} B(t)_{\gamma} - T_{\gamma} \right) \frac{d^2}{dt^2} B(t)_{\beta} + \left( B(t)_{\gamma} - T_{\gamma} \right) \frac{d}{dt} B(t)_{\beta} - 0 &= 0 \\
B(t) &= (1-t)^2 P_0 + 2(1-t)t P_1 + t^2 P_2 \\
\frac{d}{dt} B(t) &= 2t (P_2 - P_0 - 2P_1) + (2P_1 - 2P_0)
\end{align*}
\]

The coefficients of this polynomial consists of 44 terms, of which only 10 depend on the point being projected. This allows for optimization by precomputing the remaining 34 terms.

After the coefficients are calculated, the polynomial is solved analytically using Cardano’s method. The curve is sampled at all real solutions and the distance to \( T \) is calculated. The solution with the smallest distance corresponds to the projected point on the curve.

Handling junctions

The following code is an excerpt from the pixel shader that determines the final water surface normal vector.

```cpp
vec3 resultNormal; float sumFactors;
for (int i = 0; i < segmentCount; i++) {
  // Project the pixel onto segment i
  projectedPixel p = projectPixel(c1[i], c2[i], c3[i], worldPos[i], coeffConst[i]);
  // If d < width/2 add the normal vector to the result
  if (p.isVisible) {
    // If d < width/2 add the normal vector to the result
    if (d < width / 2) resultNormal += sampleNormalVector(p, lengthOffset) * factor;
    sumFactors += factor;
  }
}
vec3 resultingNormal = resultNormal / sumFactors;
```

Screenspace complexity

Because of the low vertex count, no LOD techniques are necessary for large scale river networks. Most of the computations are performed by the pixel shader, thus rendering performance depends mostly on the total surface of visible water in screenspace.

Conclusion

It is possible to use Bézier curves to efficiently model and render river networks. This technique produces convincing results of flowing water in complex environments without the use of any particle system. Because of its low complexity it is well suited for large scale river networks. Have a look at the accompanying video or demo for a better impression of the achieved results.

References

