15. Subdivision curves

Idea:
- repeatedly refine the control polygon
  \[ P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \cdots \]
- curve is the limit of an infinite process
  \[ C = \lim_{i \to \infty} P_i \]

Chaikin’s algorithm

Chakin introduced the following “corner-cutting” scheme in 1974:
- Start with a piecewise linear curve
- Insert new vertices at the midpoints (the splitting step)
- Average each vertex with the “next” neighbor (the averaging step)
- Go to the splitting step

Recommended:
Averaging masks

The limit curve is a quadratic B-spline!

Instead of averaging with the nearest neighbor, we can generalize by applying an averaging mask during the averaging step:

\[ r = (\ldots, r_3, r_0, r_1, \ldots) \]

In the case of Chaikin’s algorithm:

\[ r = \]

Lane-Riesenfeld algorithm (1980)

Use averaging masks from Pascal’s triangle:

\[ r = \frac{1}{\binom{n}{0}} \left( \begin{array}{c} n \\ n \end{array} \right) \]

Gives B-splines of degree \( n+1 \).

n=0:

n=1:

n=2:

Subdivide ad nauseum?

After each split-average step, we are closer to the limit surface.

How many steps until we reach the final (limit) position?

Can we push a vertex to its limit position without infinite subdivision? Yes!

Local subdivision matrix

Consider the cubic B-spline subdivision mask:

\[ \frac{1}{4} (1 \ 2 \ 1) \]

Now consider what happens during splitting and averaging:

We can write equations that relate points at one subdivision level to points at the previous:

\[ Q^1_L = \frac{1}{2} (Q^0_L + Q^0_R) = \frac{1}{8} (4Q^0_L + 4Q^0_R) \]

\[ Q^1 = \frac{1}{8} (Q^0_L + 6Q^0 + Q^0_R) \]

\[ Q^1_R = \frac{1}{2} (Q^0 + Q^0_R) = \frac{1}{8} (4Q^0 + 4Q^0_R) \]
**Local subdivision matrix**

We can write this as a recurrence relation in matrix form:

\[
\begin{pmatrix}
Q_l^j \\
Q_r^j \\
Q_S^j
\end{pmatrix} = \begin{pmatrix}
4 & 4 & 0 \\
1 & 6 & 1 \\
0 & 4 & 4
\end{pmatrix} \begin{pmatrix}
Q_l^{j-1} \\
Q_r^{j-1} \\
Q_S^{j-1}
\end{pmatrix}
\]

Where the \(Q\)'s are row vectors and \(S\) is the **local subdivision matrix**.

We can think about the behavior of each coordinate independently. For example, the \(x\)-coordinate:

\[
\begin{pmatrix}
x_l^j \\
x_r^j \\
x_S^j
\end{pmatrix} = \begin{pmatrix}
4 & 4 & 0 \\
1 & 6 & 1 \\
0 & 4 & 4
\end{pmatrix} \begin{pmatrix}
x_l^{j-1} \\
x_r^{j-1} \\
x_S^{j-1}
\end{pmatrix}
\]

\[
x^j = SX^{j-1}
\]

**Local subdivision matrix, cont’d**

Tracking just the \(x\) components through subdivision:

\[
X^j = SX^{j-2} = S \cdot SX^{j-3} = \ldots = S^n X^0
\]

The limit position of the \(x\)'s is then:

\[
x^\infty = \lim_{j \to \infty} S^n X^0
\]

OK, so how do we apply a matrix an infinite number of times??

**Eigenvectors and eigenvalues**

To solve this problem, we need to look at the eigenvectors and eigenvalues of \(S\). First, a review...

Let \(v\) be a vector such that:

\[
SV = \lambda v
\]

We say that \(v\) is an eigenvector with eigenvalue \(\lambda\).

An \(n \times n\) matrix can have \(n\) eigenvalues and eigenvectors:

\[
SV_1 = \lambda_1 v_1 \\
\vdots \\
SV_n = \lambda_n v_n
\]

For non-defective matrices, the eigenvectors form a basis, which means we can re-write \(X\) in terms of the eigenvectors:

\[
X = \sum_{i=1}^{n} a_i v_i
\]

**To infinity, but not beyond...**

Now let’s apply the matrix to the vector \(X\):

\[
SX = S \sum_{i=1}^{n} a_i v_i = \sum_{i=1}^{n} a_i SV_i = \sum_{i=1}^{n} a_i \lambda_i v_i
\]

Applying it \(j\) times:

\[
S^j X = S^j \sum_{i=1}^{n} a_i v_i = \sum_{i=1}^{n} a_i S^j v_i = \sum_{i=1}^{n} a_i \lambda^j_i v_i
\]

Let’s assume the eigenvalues are sorted so that:

\[
\lambda_1 > \lambda_2 > \lambda_3 \geq \ldots \geq \lambda_n
\]

Now let \(j\) go to infinity.

If \(\lambda_1 > 1\), then...

If \(\lambda_n < 1\), then...

If \(\lambda_1 = 1\), then:

\[
S^\infty X = \sum_{i=1}^{n} a_i \lambda^\infty_i v_i = a_1 v_1
\]
**Evaluation masks**

What are the eigenvalues and eigenvectors of our cubic B-spline subdivision matrix?

\[
\begin{align*}
\lambda_1 &= 1 \\
\lambda_2 &= \frac{1}{2} \\
\lambda_3 &= \frac{1}{4}
\end{align*}
\]

\[
\begin{align*}
v_1 &= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\
v_2 &= \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \\
v_3 &= \begin{pmatrix} 2 \\ -1 \\ 2 \end{pmatrix}
\end{align*}
\]

We’re OK!

But where did the x-coordinates end up?

**Evaluation masks, cont’d**

To finish up, we need to compute \( a_1 \).

It turns out that, if we call \( v \) the “right eigenvectors” then there are a corresponding set of “left eigenvectors” with the same eigenvalues such that:

\[
\begin{align*}
u_1^T S &= \lambda_1 u_1^T \\
u_n^T S &= \lambda_n u_n^T
\end{align*}
\]

Using the first left eigenvector, we can compute:

\[
x^\infty = a_1 = u_1^T x^0
\]

In fact, this works at any subdivision level:

\[
x^\infty = S^\infty x^i = u_1^T x^i
\]

The same result obtains for the y-coordinate:

\[
y^\infty = S^\infty y^i = u_1^T y^i
\]

We call \( u_1 \) an evaluation mask.

**Recipe for subdivision curves**

The evaluation mask for the cubic B-spline is:

\[
\frac{1}{6} \begin{pmatrix} 1 & 4 & 1 \end{pmatrix}
\]

Now we can cook up a simple procedure for creating subdivision curves:

- Subdivide (split+average) the control polygon a few times. Use the averaging mask.
- Push the resulting points to the limit positions. Use the evaluation mask.

Question: what is the tangent to the curve?

Answer: apply the second left eigenvector, \( u_2 \), as a tangent mask.

**DLG interpolating scheme (1987)**

Slight modification to algorithm:

- splitting step introduces midpoints
- averaging step only changes midpoints

For DLG (Dyn-Levin-Gregory), use:

\[
r = \frac{1}{16}(-2,6,10,6,-2)
\]

Since we are only changing the midpoints, the points after the averaging step do not move.
Summary

What to take home:

- How to perform the splitting and averaging steps
- What an evaluation mask is and how to use it
- An appreciation for the mathematics behind subdivision curves