BEZIER SURFACES

Bezier surfaces are constructed precisely by the same algorithm we have seen for Bezier curves. For Bezier surfaces we start by selecting a two dimensional array of control points rather than a sequence of control points as for Bezier curves.

There are a variety of ways of doing this, but in these notes we will present one of the simplest algorithms for constructing surfaces from control points. We may refer to this as the Quadrilateral Bezier Algorithm or briefly QBA. In QBA we replace the role played by segments, for Bezier curves, by quadrilaterals (4 sided Polygons).

Notice that we can construct a point inside the adjacent square by as-

ing each of its vertices $P_1, P_2, P_3, P_4$ , four non negative weights $u, v, w, z$ and constructing their center of gravity. That is the point

$$ G = uP_1 + vP_2 + wP_3 + zP_4 $$

(1)

We can easily see that this way we can obtain all points inside the square. However, since the square is a 2-dimensional object the same interior point may be obtained more than in one way. To assure that this does not happen,

we choose 2 parameters $0 \leq u \leq 1$ and $0 \leq v \leq 1$ and replace (1) by

$$ G = uP_1 + (1 - u)P_2 + (1 - u)P_3 + uP_4 $$

(2)

We also show there, the center of gravity $G$, when we choose $u = v = w = z = 1/4$, (or $u = v = 1/2$ in (2)).

Now given a $4 \times 4$ array of control points as indicated below on the left

and chosen a sequence of weights $u, v, w, z$ we can construct a new $3 \times 3$ array of control points $\|Q_{i,j}\|_{i,j=1}^3$, applying (1) to each of the squares $P_{i,j} P_{i,j+1} P_{i+1,j} P_{i+1,j+1}$, to obtain a center of gravity $Q_{i,j}$, using always the same weights $u, v, w, z$, (for $1 \leq i, j \leq 3$). Likewise we can apply the same construction to each of the squares $Q_{i,j} Q_{i+1,j} Q_{i+1,j+1}$, (for $1 \leq i, j \leq 2$) again using the same weights and obtain now a $2 \times 2$ array $R_{i,1} R_{i,2}$ $R_{2,1} R_{2,2}$.

In the final step , one single use of formula (1) gives the center gravity

$$ G(\|P_{i,j}\|_{i,j=1}^4, u, v, w, z) = uR_{1,1} + vR_{1,2} + wR_{2,1} + zR_{2,2} $$

(3)

This beautiful algorithm generalizes the construction of Bezier curves and the final formula may be viewed as the equation of the the Bezier surface determined by the $4 \times 4$ initial array $\|P_{i,j}\|_{i,j=1}^4$. Clearly this can equally be carried out starting with any $n \times n$ initial array $\|P_{i,j}\|_{i,j=1}^n$. We have implemented this algorithm in Mathematica using the following two procedures

quadra[M_, u_, v_, w_, z_] := Block[{i, j, ut, n},
   (n = Length[M];
   ut = Table[uM[[i]][[j]] + vM[[i]][[j + 1]] + wM[[i + 1]][[j + 1]] + zM[[i + 1]][[j]], {i, 1, n - 1}, {j, 1, n - 1}];
   Return[ut])]

Where $M$ is the given initial array of control points. This procedure starts with an $n \times n$ array and delivers an $(n - 11) \times (n - 11)$ array.
\begin{verbatim}
BezSurf[M_, u_, v_, w_, z_] := Block[{S, i, prev, new},
  (n = Length[M];
   S[1] = M;
   prev = M;
   Do[new = quadra[prev, u, v, w, z];
     prev = new;
     S[i] = new, {i, 1, n - 2}];
   S[n] = FinPt[prev, u, v, w, z];
   Return[Expand[S[n]]])]
where FinPt is the procedure that implements formula (3), That is

\text{FinPt}[C_, u_, v_, w_, z_] := uC[1][[1]] + vC[1][[2]] + wC[2][[2]] + zC[2][[1]]

Calling \text{BezSurf} with the following 4 × 4 array of control points

\begin{align}
\text{surf1} = \{(1, 1, 5), (1, 2, 3), (1, 3, 3), (1, 4, 5), \\
(2, 1, 3), (2, 2, 1), (2, 3, 1), (2, 4, 3), \\
(3, 1, 3), (3, 2, 1), (3, 3, 1), (3, 4, 3), \\
(4, 1, 5), (4, 2, 3), (4, 3, 3), (4, 4, 5)\}
\end{align}

and

\[(u, v, w, z) = (uv, v(1 - u), (1 - v)(1 - u), (1 - v)u)\]

We obtained the following surprisingly simple parametric equations of the resulting surface

\[\text{bingo} = \{4 - 3v, 4 - 3u, 5 - 6u + 6u^2 - 6v\}\]

This given, the procedure

\text{uvPPO3}[surfaces_, a_, b_, c_] :=
  param[surfaces, u, 0, 1, v, 0, 1, 
  AspectRatio -> 1, ViewPoint -> a, b, c, Axes -> True, 
  PlotPoints -> 40] /. param -> ParametricPlot3D

delivered the attached Bezier surface. It will you good to find out why
the control points in (4) caused the surface to look like this.

The actual command was

\text{uvPPO3}[bingo, 3, 3, 4]

The next suggestion that leads to beautiful displays is to obtain your \(n \times n\) array of control points by the
random number generator. The following two procedures implement this idea.

\text{ra}[a_] := RandomInteger[a]

\text{mkrasurf}[n_, a_] := Table[Table[ra[a], ra[a], ra[a], n], n]
A single call of

\[ \text{rando} = \text{mkrasurf}[4,5] \]

using the following more fancy display procedure

\[
\text{uvfancyPP3[surfaces_, color_] :=}
\text{ParametricPlot3D}\left[ \text{surfaces}, u, 0, 1, v, 0, 1, \text{AspectRatio} -> 1,\right.
\text{Boxed} -> \text{True},
\text{PlotStyle} -> \text{color}, \text{Specularity[White, 200]}, \text{Opacity[0.9]},
\text{Axes} -> \text{False}, \text{PlotRange} -> \text{All}, \text{Mesh} -> \text{None}\right]
\]

yielded me the following display

Not being terribly impressed by the result I decided to combine the Kaleidoscope procedure with the Bezier Surface procedure. The idea is very easy to implement, All you have to do is feed your surface parametric equation 
\[ S[u, v] = \{a(u, v), b(u, v), c(u, v)\} \]
into a slightly modified kaleidoscope procedure, which replaces the sequence
\[
\{ \{a, b\}, \{a, -b\}, \{-a, b\}, \{-a, -b\}, \{b, a\}, \{b, -a\}, \{-b, a\}, \{-b, -a\} \}
\]
by the sequence
\[
\{ \{a, b, c\}, \{a, -b, c\}, \{-a, b, c\}, \{-a, -b, c\}, \{b, a, c\}, \{b, -a, c\}, \{-b, a, c\}, \{-b, -f, c\} \}.
\]
This will produce 6 surfaces which are the reflections of the original surface across planes that contain the z-axis. I fed that \textbf{rando} surface above translated by the vector \(-2,3,0\) (no point translating in the z-axis direction) and got the following beautiful \textit{kaleysurface}

Play with these Bezier surfaces and have fun