The Origins of the Teapot

Frank Crow, Xerox PARC (from conversations with Martin Newell and Jim Blinn)

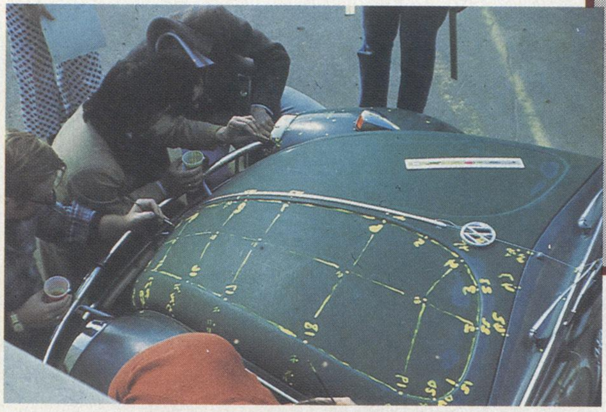


Figure 1. Digitizing circa 1971.

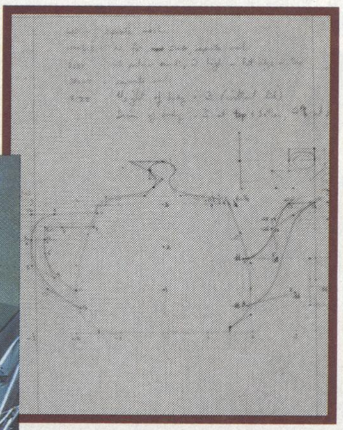


Figure 2. The original sketch of the Teapot.

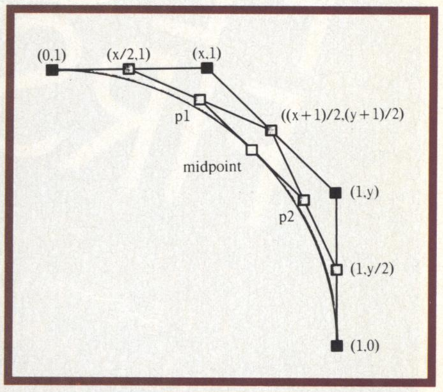


Figure 3. Bezier approximation to a quarter-circle.

Are you interested in more articles of this type? Are you interested in data (such as the teapot data) or code or both? Do you have public domain data and/or code that you would be willing to share? Please send your answers and comments to Chip Hatfield, Lawrence Livermore National Laboratory, PO Box 5504, L-156, Livermore, CA 94550 or HATFIELD @LLL-ICDC.ARPA.

In the early 1970s at the University of Utah, there was substantial activity in the development of rendering algorithms. However, there was a constant shortage of data for interesting shapes to be displayed with these algorithms. Tiring quickly of spheres, cubes, tubes, and other easily generated shapes, interesting efforts were often mounted to capture more elaborate data.

One set of efforts took the form of developing automated methods for capturing physical measurements. 3-D digitizers using mechanical means, photogrammetry, and even lasers were developed. However, much of the interesting data was completely handcrafted. In 1971 Ivan Sutherland had his computer graphics class digitize his VW beetle (see Figure 1). This took weeks, during which time the VW could be seen here and there in Salt Lake City covered with paint used for reference marks.

In a similar spirit, Martin Newell, in 1975, noticed the pleasing shape of the teapot he kept on his desk. Taking some square-grid graph paper, he made a rough sketch of the profile of the teapot as seen from the side (see Figure 2), capturing the essence of the shape, but not the precise dimensions (differences can be seen around the spout and the knob on the lid). From the sketch he guessed at the location of suitable control points for cubic Bezier splines,[1,2] and then measured these points using the graph paper grid.

Having the control points for a profile of the teapot, it remained to develop a three-dimensional surface. The lid, rim, and body of the teapot were treated as surfaces of revolution, while the spout and handle were developed as three-dimensional tubes.

From Figure 2 it can be seen that the body and rim are formed from three Bezier segments sketched along the right side of the profile. Four points are needed to define a cubic segment. The rim is defined by points forming a leftward-leaning trapezoid sitting above the horizontal gridline marked "3." The upper body is defined by four points hugging the profile while running from gridline 3 to gridline 1. The lower body is then defined by the remaining four points running down to the bottom.

Once you have a set of Bezier control points which generate a close approximation to a circle, a surface of revolution can be built by replicating the circle, scaled and translated, for each of the control points on the profile. For the teapot this produces a mesh of control points with each vertical path containing the control points for three Bezier segments and each horizon-

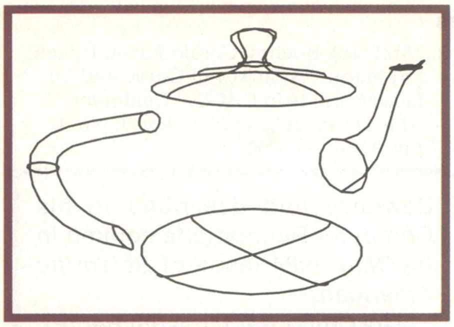


Figure 4. Teapot patch boundaries (rim and body removed for clarity).

tal path containing four Bezier segments, each approximating a quarter-circle. The patch structure of the surface is then implicit in that patch corners occur where a control point lies on segment boundaries on both vertical and horizontal paths.

To approximate a quarter-circle, we know that the polygon formed by the four control points for a Bezier cubic (the control polygon) must obey some constraints (see Figure 3). It must be symmetric about its midpoint. The line segments at the ends must be at right angles to one another. Finally, the midpoint of the resulting curve should lie on the circle defined by the endpoints and the slopes at the endpoints. From these constraints the coordinates of the inner knots can be derived.

Let's look at the quarter-circle from (0,1) to (1,0). Its midpoint lies at (sqrt(2)/2, sqrt(2)/2). Using the standard de Casteljau subdivision construction[3] to find the mid point of the curve, we can work back to find the control points that will give us the right midpoint. Recall that the midpoint of the curve is the midpoint of the line, p1-p2. p1 and p2 are themselves constructed from the midpoints of lines connecting midpoints of edges of the control polygon.

We know that the first and last edges of the control polygon must be perpendicular to one another. Since the polygon must also be symmetric, we can find the control point coordinates as follows:

The four control points must be (0,1), (x,1), (1,y), (1,0), where x = y. Let's look just at the x coordinate.

Working backwards from the midpoint:

sqrt(2)/2 = (p1x+p2x)/2,

$p1_x$ = (x/2+(x+1)/2)/2 = (2x+1)/4,

and

$p2_x$ = ((x+1)/2+1)/2 = (x+3)/4.

Substituting gives:

sqrt(2)/2 = ($p1_x$+$p2_x$)/2 = (3+4)/8.

Then, solving for x:

x = 4/3(sqrt(2)-1),

or roughly 0.5523.

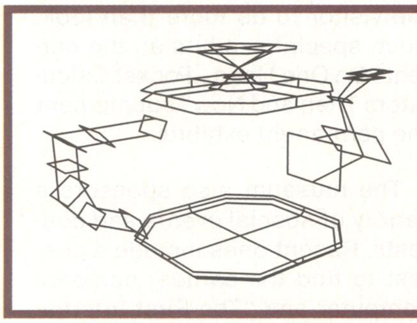


Figure 5. Teapot control points (rim and body removed for clarity).

The spout and handle, each formed from four patches, were made by guessing at the positions of control points to form the inner and outer profiles as seen in Figure 2 The x and y coordinates of the control points were then digitized from the sketch. For the handle, a second set of profiles was specified by providing a near and a far z-coordinate for each digitized point. The pairs of profiles are defined by a constant displacement as can be seen in Figure 5. The control points for the spout were similarly defined, except that the displacement was varied to allow a tapering shape. In Figure 2 a top-view sketch of the spout can be seen above its profile sketch. This formed the basis for the displacements in z. The top view also shows how the tip of the spout is turned in upon itself.

Once the data were generated, a line-drawing program, similar to the code provided in this issue, was used to display the resulting patches on an early storage tube display.

The teapot itself, and the original

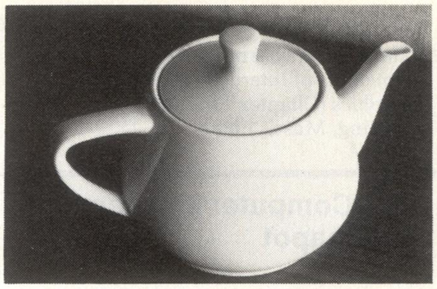


Figure 6. The original teapot from Newell's desk.

data, are somewhat taller than the shape we recognize today. As can be seen from notes on the original sketch, the original data showed the bottom of the rim to be three units above the origin. The rim in the current data is 2.4 units above the origin, or 80 percent as high. The original data was scaled a number of times in its early life. Jim Blinn recalls that the current shape was felt to be the most esthetic at some point. Figure 6, an illustration from Martin Newell's PhD dissertation, shows the teapot in its original form.

The teapot's first major appearance was in a paper at SIGGRAPH 76 (reprinted in the October 1976 issue of the *Communications of the ACM*). Much of the teapot's fame comes from this use and subsequent, ever more spectacular, images by Jim Blinn. Blinn found the shape to be a particularly good test object. It has both positive and negative curvature on the surface. The neck of the knob on the lid is a saddle shape, a particularly difficult form for some algorithms to render. Note also that the handle and spout both intersect the surface of the body, forcing intersecting surfaces to be handled properly.

The original data had only 28 patches. The additional four patches in the data used here add a bottom to the teapot. Since the real teapot served primarily to hold water, it seemed only proper that its numerically defined analog should be equally capable.

Martin Newell's teapot now resides in the Computer Museum in Boston, where it is displayed next to its computer-generated likeness. It is unlikely that it will ever again need to hold water.

References on Bezier curves and surfaces

1. James D. Foley, Andries van Dam. *Fundamentals of Interactive Computer Graphics* (Chapter 13). Addison-Wesley, Reading, Mass., 1982.
2. William M. Newman, Robert F. Sproull. *Principles of Interactive Computer Graphics*, 2nd Ed. (Chapter 21). McGraw-Hill, New York, 1973.
3. Wolfgang Boehm, Gerald Farin, Jurgen Kahmann. "A Survey of Curve and Surface Methods in CAGD," *Computer-Aided Geometric Design*, Vol. 1, No. 1. July 1984, pp. 1-60.

The Computer Museum and the Teapot

The original ceramic teapot that inspired Martin Newell's computer creation is now on display at The Computer Museum in Boston, Massachusetts, where it is in a prominent display in one of the four galleries that showcase the most extensive collection of historically significant computer technology ever assembled. These galleries focus on "The Vacuum Tube Era," "The Transistor Era," "The Integrated Circuit Era," and "The Computer and the Image."

The museum began with the effort to rescue the Whirlwind—the world's first real-time, parallel, vacuum tube computer with a core memory. This led to the opening of the museum in 1979 by Digital Equipment Corporation in its Marlboro, Massachusetts, facility. The museum became an independent nonprofit institution in 1982 and moved to its present facilities on Museum Wharf in 1984.

Today, the visitor can wander through the museum and see a collection of photomurals, timelines, and vintage computers installed in re-creations of their original setting. Dozens of interactive exhibits, including a gallery of personal computers and state-of-the-art graphics display processors, invite the visitor to do more than look. Such special exhibits as the current "On One Hand...Pocket Calculators Then and Now" supplement the permanent exhibits.

The museum also sponsors a variety of special events and contests. Recent ones include a contest to find the earliest personal computer and "The First International Core War Tournament." The personal computer contest brought 137 new machines to the museum's collection, including one thought to be the first commercial personal computer, the Kenback-1 (1971). In the tournament 29 programs paired off in battle for supremacy within a computer in the game created by A.K. Dewdney and described in his Computer Recreations column in the May 1984 issue of *Scientific American.*

The Computer Museum Report, a quarterly publication, provides fascinating and informal looks into the history of computing as well as details of activities at the museum. In 1986 the report carried articles by such distinguished authors as J. Presper Eckert on the genesis of the Eniac, Daniel Bricklin on the origins of VisiCalc, and Stephen Wozniack on his career in electronics and the development of the Apple 1.

The Computer Museum is located at Museum Wharf, 300 Congress Street, Boston. Its hours are 10 a.m. to 6 p.m. Tuesday through Sunday and until 9 p.m. on Friday. Membership in The Computer Museum brings you *The Computer Museum Report*, free admission, invitations to previews, and a 10 percent discount at the museum's store. Write to the Membership Coordinator or call (617) 426-2800.

Pascal Code to Display Wire Frame Model

```
{ The following Pascal procedure, "Display_Patches," will draw a wire frame representation of the Bézier patch data in
the arrays "Ducks" and "Patches." The loading of the data into the arrays is not shown. The following global declarations
are assumed: }

CONST
Degree       = 3 ;   { The degree of the Bézier spline used }
Duck_Count   = 306 ; { The number of control points, "Ducks" }
Patch_Count  = 32 ;  { The number of surface patches }

TYPE
Duck_Type         = RECORD X , Y , Z : Real ; END ;   { Each duck is a three-vector }
Duck_Index_Type   = 1 .. Duck_Count ;
Duck_Array_Type   = ARRAY [ 1 .. Duck_Count ] OF Duck_Type ;
Patch_Type        = ARRAY [ 0 .. Degree , 0 .. Degree ] OF Duck_Index_Type ;   { Each patch points to 16 ducks }
Patch_Array_Type  = ARRAY [ 1 .. Patch_Count ] OF Patch_Type ;

VAR
Ducks     : Duck_Array_Type ;   { store ducks here }
Patches   : Patch_Array_Type ;  { which ducks go with which patches }

{ The procedure "Display_Patches" and its support procedures are given below. Note that the parameter "Steps" controls
the granularity of the subdivision of each patch for display. Try a value of 6 to start, then experiment. The procedures
"Move" and "Draw" should be replaced with the graphics procedures appropriate for your system. }
```

```
PROCEDURE Blend_Vector (D0 , D1 , D2 , D3 : Duck_Type ; T : Real ; VAR Result : Duck_Type) ;

BEGIN { Calculate vector cubic Bézier spline value at parameter T }
Result.X := D0.X*(1-T)*(1-T)*(1-T) + D1.X*3*T*(1-T)*(1-T) + D2.X*3*T*T*(1-T) + D3.X*T*T*T ;
Result.Y := D0.Y*(1-T)*(1-T)*(1-T) + D1.Y*3*T*(1-T)*(1-T) + D2.Y*3*T*T*(1-T) + D3.Y*T*T*T ;
Result.Z := D0.Z*(1-T)*(1-T)*(1-T) + D1.Z*3*T*(1-T)*(1-T) + D2.Z*3*T*T*(1-T) + D3.Z*T*T*T ;
END ; { procedure blend_vector }
```

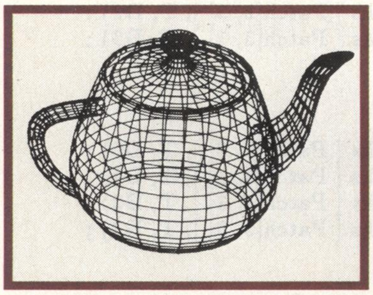
```
PROCEDURE Display_Curve (D0, D1, D2, D3 : Duck_Type ; Steps : Integer) ;

VAR                     { Find "Steps+1" points on the spline and }
T, Step  : Real ;        { draw "Steps" line segments }
Temp     : Duck_Type ;

BEGIN
Step := 1 / Steps ;
T := Step ;
Move (D0.X, D0.Y, D0.Z) ;              { move to start of spline }
WHILE T < 1 + Step / 2 DO
  BEGIN
  Blend_Vector (D0, D1, D2, D3, T, Temp) ;
  Draw (Temp.X, Temp.Y, Temp.Z) ;    { draw line segment to next point }
  T := T + Step ;
  END ; { while t < }
END ; { procedure display_curve }
```

```
PROCEDURE Display_Patch (VAR Patch : Patch_Type ; Steps : Integer) ;

VAR
T    : Real ;
Step : Real ;
D0, D1, D2, D3 : Duck_Type ;    { ducks for a particular constant U or V value }

BEGIN
Step := 1 / Steps ;
T := 0 ;
WHILE T < 1 + Step / 2 DO
  BEGIN                  { splines of constant U }

   Blend_Vector (Ducks [ Patch[0, 0] ], Ducks [ Patch[0, 1] ], Ducks [ Patch[0, 2] ], Ducks [ Patch[0, 3] ], T, D0) ;
   Blend_Vector (Ducks [ Patch[1, 0] ], Ducks [ Patch[1, 1] ], Ducks [ Patch[1, 2] ], Ducks [ Patch[1, 3] ], T, D1) ;
   Blend_Vector (Ducks [ Patch[2, 0] ], Ducks [ Patch[2, 1] ], Ducks [ Patch[2, 2] ], Ducks [ Patch[2, 3] ], T, D2) ;
   Blend_Vector (Ducks [ Patch[3, 0] ], Ducks [ Patch[3, 1] ], Ducks [ Patch[3, 2] ], Ducks [ Patch[3, 3] ], T, D3) ;

   Display_Curve (D0, D1, D2, D3, Steps) ;

                       { splines of constant V }
   Blend_Vector (Ducks [ Patch[0, 0] ], Ducks [ Patch[1, 0] ], Ducks [ Patch[2, 0] ], Ducks [ Patch[3, 0] ], T, D0) ;
   Blend_Vector (Ducks [ Patch[0, 1] ], Ducks [ Patch[1, 1] ], Ducks [ Patch[2, 1] ], Ducks [ Patch[3, 1] ], T, D1) ;
   Blend_Vector (Ducks [ Patch[0, 2] ], Ducks [ Patch[1, 2] ], Ducks [ Patch[2, 2] ], Ducks [ Patch[3, 2] ], T, D2) ;
   Blend_Vector (Ducks [ Patch[0, 3] ], Ducks [ Patch[1, 3] ], Ducks [ Patch[2, 3] ], Ducks [ Patch[3, 3] ], T, D3) ;

   Display_Curve (D0, D1, D2, D3, Steps) ;
   T := T + Step ;
   END ; { while t < }
 END ; { procedure display_patch }
```

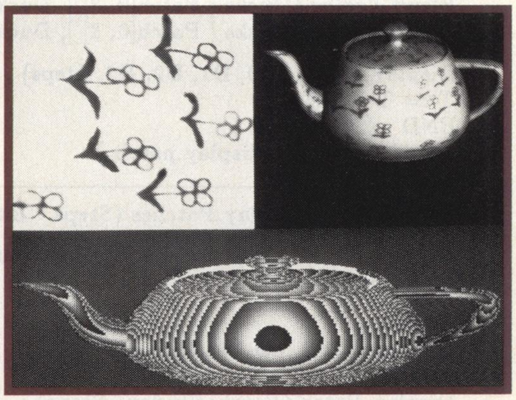
```
PROCEDURE Display_Patches (Steps : Integer) ;

VAR                    { "Steps" tells how much to divide up the patches }
Index : Integer ;

BEGIN
FOR Index := 1 TO Patch_Count DO
  BEGIN
  Display_Patch (Patches[Index], Steps) ;
  END ; { for index }
END ; { procedure display_patches } { by Charles W. Grant, Lawrence Livermore National Laboratory }
```

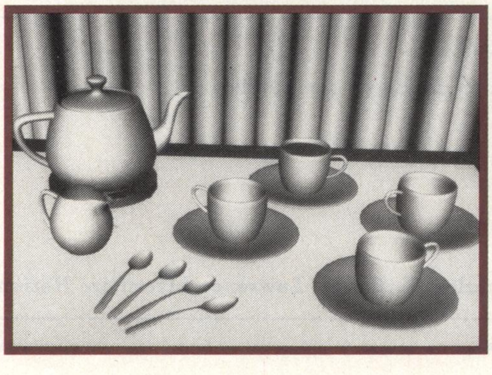
Teapot Data – Patch Definitions

Patch Number	16 Indices Into Table of x,y,z Values															
rim –																
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	4	17	18	19	8	20	21	22	12	23	24	25	16	26	27	28
3	19	29	30	31	22	32	33	34	25	35	36	37	28	38	39	40
4	31	41	42	1	34	43	44	5	37	45	46	9	40	47	48	13
body –																
5	13	14	15	16	49	50	51	52	53	54	55	56	57	58	59	60
6	16	26	27	28	52	61	62	63	56	64	65	66	60	67	68	69
7	28	38	39	40	63	70	71	72	66	73	74	75	69	76	77	78
8	40	47	48	13	72	79	80	49	75	81	82	53	78	83	84	57
9	57	58	59	60	85	86	87	88	89	90	91	92	93	94	95	96
10	60	67	68	69	88	97	98	99	92	100	101	102	96	103	104	105
11	69	76	77	78	99	106	107	108	102	109	110	111	105	112	113	114
12	78	83	84	57	108	115	116	85	111	117	118	89	114	119	120	93
handle –																
13	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136
14	124	137	138	121	128	139	140	125	132	141	142	129	136	143	144	133
15	133	134	135	136	145	146	147	148	149	150	151	152	69	153	154	155
16	136	143	144	133	148	156	157	145	152	158	159	149	155	160	161	69
spout –																
17	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177
18	165	178	179	162	169	180	181	166	173	182	183	170	177	184	185	174
19	174	175	176	177	186	187	188	189	190	191	192	193	194	195	196	197
20	177	184	185	174	189	198	199	186	193	200	201	190	197	202	203	194
lid –																
21	204	204	204	204	207	208	209	210	211	211	211	211	212	213	214	215
22	204	204	204	204	210	217	218	219	211	211	211	211	215	220	221	222
23	204	204	204	204	219	224	225	226	211	211	211	211	222	227	228	229
24	204	204	204	204	226	230	231	207	211	211	211	211	229	232	233	212
25	212	213	214	215	234	235	236	237	238	239	240	241	242	243	244	245
26	215	220	221	222	237	246	247	248	241	249	250	251	245	252	253	254
27	222	227	228	229	248	255	256	257	251	258	259	260	254	261	262	263
28	229	232	233	212	257	264	265	234	260	266	267	238	263	268	269	242
bottom –																
29	270	270	270	270	279	280	281	282	275	276	277	278	271	272	273	274
30	270	270	270	270	282	289	290	291	278	286	287	288	274	283	284	285
31	270	270	270	270	291	298	299	300	288	295	296	297	285	292	293	294
32	270	270	270	270	300	305	306	279	297	303	304	275	294	301	302	271


Martin Newell's original wireframe teapot.

Shaded texture mapped teapot. Texture map was generated by Fourier synthesis.


These three images are (a) texture map, (b) Teapot with environment map, and (c) texture mapped on Z-buffer at bottom of screen, 2 bytes/pixel so twice as wide.

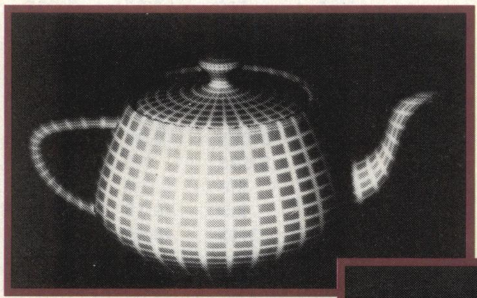

Tea scene from Martin Newell's PhD thesis, University of Utah, 1974. Environment mapped teapot.

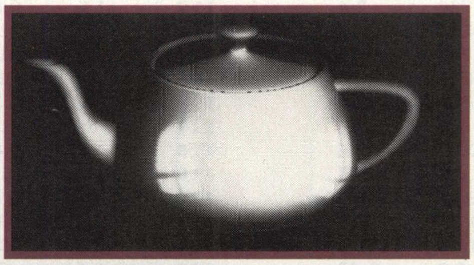
Teapot Data – Vertices

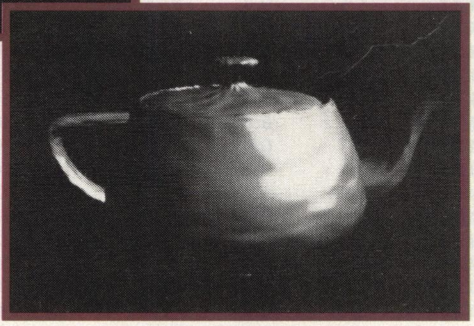
Vertex Number	*x* Value	*y* Value	*z* Value
1	1.4	0.0	2.4
2	1.4	-0.784	2.4
3	0.784	-1.4	2.4
4	0.0	-1.4	2.4
5	1.3375	0.0	2.53125
6	1.3375	-0.749	2.53125
7	0.749	-1.3375	2.53125
8	0.0	-1.3375	2.53125
9	1.4375	0.0	2.53125
10	1.4375	-0.805	2.53125
11	0.805	-1.4375	2.53125
12	0.0	-1.4375	2.53125
13	1.5	0.0	2.4
14	1.5	-0.84	2.4
15	0.84	-1.5	2.4
16	0.0	-1.5	2.4
17	-0.784	-1.4	2.4
18	-1.4	-0.784	2.4
19	-1.4	0.0	2.4
20	-0.749	-1.3375	2.53125
21	-1.3375	-0.749	2.53125
22	-1.3375	0.0	2.53125
23	-0.805	-1.4375	2.53125
24	-1.4375	-0.805	2.53125
25	-1.4375	0.0	2.53125
26	-0.84	-1.5	2.4
27	-1.5	-0.84	2.4
28	-1.5	0.0	2.4
29	-1.4	0.784	2.4
30	-0.784	1.4	2.4
31	0.0	1.4	2.4
32	-1.3375	0.749	2.53125
33	-0.749	1.3375	2.53125
34	0.0	1.3375	2.53125
35	-1.4375	0.805	2.53125
36	-0.805	1.4375	2.53125
37	0.0	1.4375	2.53125
38	-1.5	0.84	2.4
39	-0.84	1.5	2.4
40	0.0	1.5	2.4
41	0.784	1.4	2.4
42	1.4	0.784	2.4
43	0.749	1.3375	2.53125
44	1.3375	0.749	2.53125
45	0.805	1.4375	2.53125
46	1.4375	0.805	2.53125
47	0.84	1.5	2.4
48	1.5	0.84	2.4
49	1.75	0.0	1.875
50	1.75	-0.98	1.875
51	0.98	-1.75	1.875
52	0.0	-1.75	1.875
53	2.0	0.0	1.35
54	2.0	-1.12	1.35
55	1.12	-2.0	1.35
56	0.0	-2.0	1.35
57	2.0	0.0	0.9
58	2.0	-1.12	0.9
59	1.12	-2.0	0.9
60	0.0	-2.0	0.9
61	-0.98	-1.75	1.875
62	-1.75	-0.98	1.875
63	-1.75	0.0	1.875
64	-1.12	-2.0	1.35
65	-2.0	-1.12	1.35
66	-2.0	0.0	1.35
67	-1.12	-2.0	0.9
68	-2.0	-1.12	0.9
69	-2.0	0.0	0.9
70	-1.75	0.98	1.875
71	-0.98	1.75	1.875
72	0.0	1.75	1.875
73	-2.0	1.12	1.35
74	-1.12	2.0	1.35
75	0.0	2.0	1.35
76	-2.0	1.12	0.9
77	-1.12	2.0	0.9
78	0.0	2.0	0.9
79	0.98	1.75	1.875
80	1.75	0.98	1.875
81	1.12	2.0	1.35
82	2.0	1.12	1.35
83	1.12	2.0	0.9
84	2.0	1.12	0.9
85	2.0	0.0	0.45
86	2.0	-1.12	0.45
87	1.12	-2.0	0.45
88	0.0	-2.0	0.45
89	1.5	0.0	0.225
90	1.5	-0.84	0.225
91	0.84	-1.5	0.225
92	0.0	-1.5	0.225
93	1.5	0.0	0.15
94	1.5	-0.84	0.15
95	0.84	-1.5	0.15
96	0.0	-1.5	0.15
97	-1.12	-2.0	0.45
98	-2.0	-1.12	0.45
99	-2.0	0.0	0.45
100	-0.84	-1.5	0.225
101	-1.5	-0.84	0.225
102	-1.5	0.0	0.225
103	-0.84	-1.5	0.15
104	-1.5	-0.84	0.15
105	-1.5	0.0	0.15
106	-2.0	1.12	0.45
107	-1.12	2.0	0.45
108	0.0	2.0	0.45
109	-1.5	0.84	0.225
110	-0.84	1.5	0.225
111	0.0	1.5	0.225
112	-1.5	0.84	0.15
113	-0.84	1.5	0.15
114	0.0	1.5	0.15
115	1.12	2.0	0.45
116	2.0	1.12	0.45
117	0.84	1.5	0.225
118	1.5	0.84	0.225
119	0.84	1.5	0.15
120	1.5	0.84	0.15
121	-1.6	0.0	2.025
122	-1.6	-0.3	2.025
123	-1.5	-0.3	2.25
124	-1.5	0.0	2.25
125	-2.3	0.0	2.025
126	-2.3	-0.3	2.025
127	-2.5	-0.3	2.25
128	-2.5	0.0	2.25
129	-2.7	0.0	2.025
130	-2.7	-0.3	2.025
131	-3.0	-0.3	2.25
132	-3.0	0.0	2.25
133	-2.7	0.0	1.8
134	-2.7	-0.3	1.8
135	-3.0	-0.3	1.8
136	-3.0	0.0	1.8
137	-1.5	0.3	2.25
138	-1.6	0.3	2.025
139	-2.5	0.3	2.25
140	-2.3	0.3	2.025
141	-3.0	0.3	2.25
142	-2.7	0.3	2.025
143	-3.0	0.3	1.8
144	-2.7	0.3	1.8
145	-2.7	0.0	1.575
146	-2.7	-0.3	1.575
147	-3.0	-0.3	1.35
148	-3.0	0.0	1.35
149	-2.5	0.0	1.125
150	-2.5	-0.3	1.125
151	-2.65	-0.3	0.9375
152	-2.65	0.0	0.9375
153	-2.0	-0.3	0.9
154	-1.9	-0.3	0.6
155	-1.9	0.0	0.6
156	-3.0	0.3	1.35
157	-2.7	0.3	1.575
158	-2.65	0.3	0.9375
159	-2.5	0.3	1.125
160	-1.9	0.3	0.6
161	-2.0	0.3	0.9
162	1.7	0.0	1.425
163	1.7	-0.66	1.425
164	1.7	-0.66	0.6
165	1.7	0.0	0.6
166	2.6	0.0	1.425
167	2.6	-0.66	1.425
168	3.1	-0.66	0.825
169	3.1	0.0	0.825
170	2.3	0.0	2.1
171	2.3	-0.25	2.1
172	2.4	-0.25	2.025
173	2.4	0.0	2.025
174	2.7	0.0	2.4
175	2.7	-0.25	2.4
176	3.3	-0.25	2.4
177	3.3	0.0	2.4
178	1.7	0.66	0.6
179	1.7	0.66	1.425
180	3.1	0.66	0.825
181	2.6	0.66	1.425
182	2.4	0.25	2.025
183	2.3	0.25	2.1
184	3.3	0.25	2.4
185	2.7	0.25	2.4
186	2.8	0.0	2.475
187	2.8	-0.25	2.475
188	3.525	-0.25	2.49375
189	3.525	0.0	2.49375
190	2.9	0.0	2.475
191	2.9	-0.15	2.475
192	3.45	-0.15	2.5125
193	3.45	0.0	2.5125
194	2.8	0.0	2.4
195	2.8	-0.15	2.4
196	3.2	-0.15	2.4
197	3.2	0.0	2.4
198	3.525	0.25	2.49375
199	2.8	0.25	2.475
200	3.45	0.15	2.5125
201	2.9	0.15	2.475
202	3.2	0.15	2.4
203	2.8	0.15	2.4
204	0.0	0.0	3.15
205	0.0	-0.002	3.15
206	0.002	0.0	3.15
207	0.8	0.0	3.15
208	0.8	-0.45	3.15
209	0.45	-0.8	3.15
210	0.0	-0.8	3.15

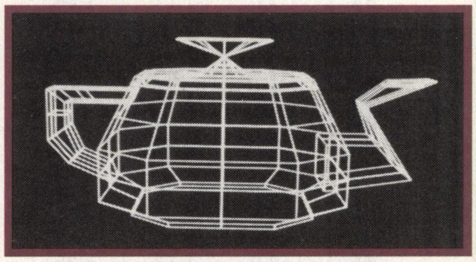
Vertex Number	*x* Value	*y* Value	*z* Value
211	0.0	0.0	2.85
212	0.2	0.0	2.7
213	0.2	-0.112	2.7
214	0.112	-0.2	2.7
215	0.0	-0.2	2.7
216	-0.002	0.0	3.15
217	-0.45	-0.8	3.15
218	-0.8	-0.45	3.15
219	-0.8	0.0	3.15
220	-0.112	-0.2	2.7
221	-0.2	-0.112	2.7
222	-0.2	0.0	2.7
223	0.0	0.002	3.15
224	-0.8	0.45	3.15
225	-0.45	0.8	3.15
226	0.0	0.8	3.15
227	-0.2	0.112	2.7
228	-0.112	0.2	2.7
229	0.0	0.2	2.7
230	0.45	0.8	3.15
231	0.8	0.45	3.15
232	0.112	0.2	2.7
233	0.2	0.112	2.7
234	0.4	0.0	2.55
235	0.4	-0.224	2.55
236	0.224	-0.4	2.55
237	0.0	-0.4	2.55
238	1.3	0.0	2.55
239	1.3	-0.728	2.55
240	0.728	-1.3	2.55
241	0.0	-1.3	2.55
242	1.3	0.0	2.4
243	1.3	-0.728	2.4
244	0.728	-1.3	2.4
245	0.0	-1.3	2.4
246	-0.224	-0.4	2.55
247	-0.4	-0.224	2.55
248	-0.4	0.0	2.55
249	-0.728	-1.3	2.55
250	-1.3	-0.728	2.55
251	-1.3	0.0	2.55
252	-0.728	-1.3	2.4
253	-1.3	-0.728	2.4
254	-1.3	0.0	2.4
255	-0.4	0.224	2.55
256	-0.224	0.4	2.55
257	0.0	0.4	2.55
258	-1.3	0.728	2.55
259	-0.728	1.3	2.55
260	0.0	1.3	2.55
261	-1.3	0.728	2.4
262	-0.728	1.3	2.4
263	0.0	1.3	2.4
264	0.224	0.4	2.55
265	0.4	0.224	2.55
266	0.728	1.3	2.55
267	1.3	0.728	2.55
268	0.728	1.3	2.4
269	1.3	0.728	2.4
270	0.0	0.0	0.0
271	1.5	0.0	0.15
272	1.5	0.84	0.15
273	0.84	1.5	0.15
274	0.0	1.5	0.15
275	1.5	0.0	0.075
276	1.5	0.84	0.075
277	0.84	1.5	0.075
278	0.0	1.5	0.075
279	1.425	0.0	0.0
280	1.425	0.798	0.0
281	0.798	1.425	0.0
282	0.0	1.425	0.0
283	-0.84	1.5	0.15
284	-1.5	0.84	0.15
285	-1.5	0.0	0.15
286	-0.84	1.5	0.075
287	-1.5	0.84	0.075
288	-1.5	0.0	0.075
289	-0.798	1.425	0.0
290	-1.425	0.798	0.0
291	-1.425	0.0	0.0
292	-1.5	-0.84	0.15
293	-0.84	-1.5	0.15
294	0.0	-1.5	0.15
295	-1.5	-0.84	0.075
296	-0.84	-1.5	0.075
297	0.0	-1.5	0.075
298	-1.425	-0.798	0.0
299	-0.798	-1.425	0.0
300	0.0	-1.425	0.0
301	0.84	-1.5	0.15
302	1.5	-0.84	0.15
303	0.84	-1.5	0.075
304	1.5	-0.84	0.075
305	0.798	-1.425	0.0
306	1.425	-0.798	0.0

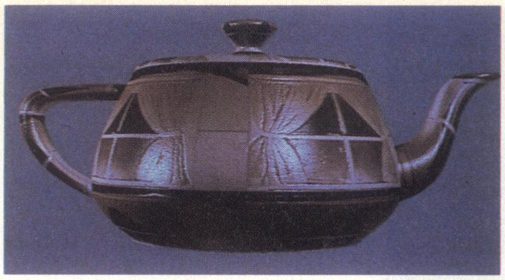
Note that the data for the 306 vertices needs to be stored in the array "Ducks" and the data for the 32 patches in the array "Patches." The "Vertex Numbers" and the "Patch Numbers" printed in the tables are NOT to be stored in these arrays. These numbers are given to assist visual inspection and interpretation of the data only.

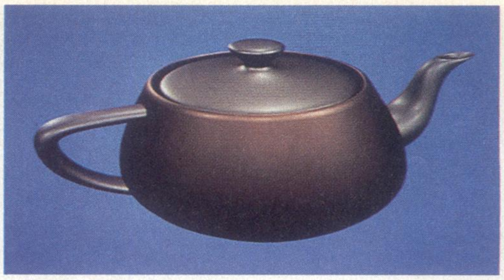

Texture map with painted grid picture.

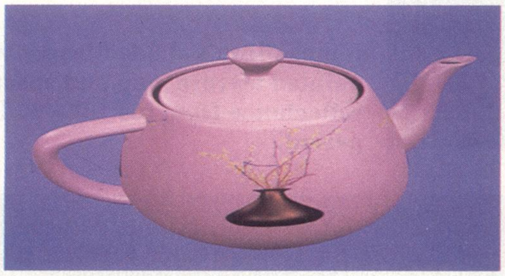

Environment mapped teapot.


Bump-mapped teapot.


Map of control points for teapot.

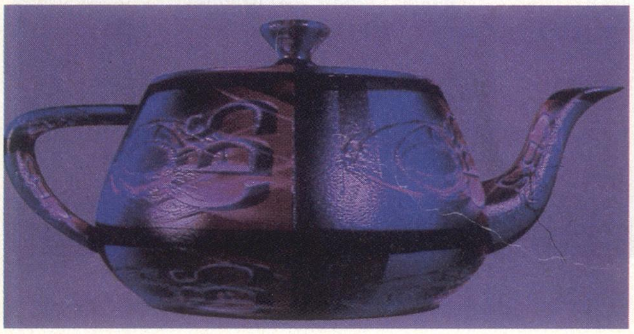

a

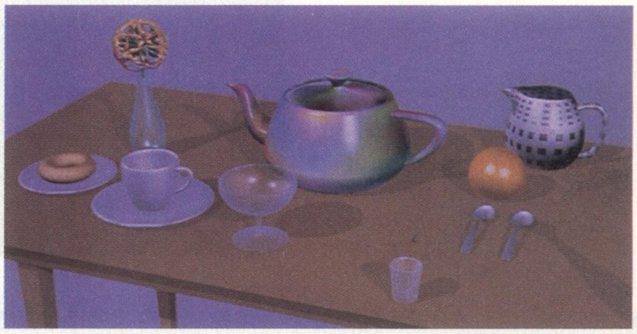

b


c

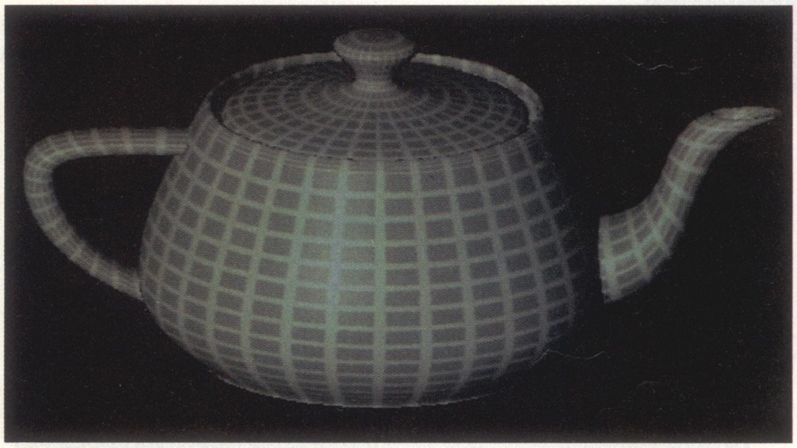
d

Kindly supplied to *CG&A* by The Computer Museum, these pictures are (a) owned by Omnibus Computer Graphics, (b) a picture done by Rob Cook when he was at Lucasfilm in 1982, (c) another Robert Cook image with a plant by Alvy Ray Smith when he was at Lucasfilm, and (d) a picture showing work by Martha Everson.

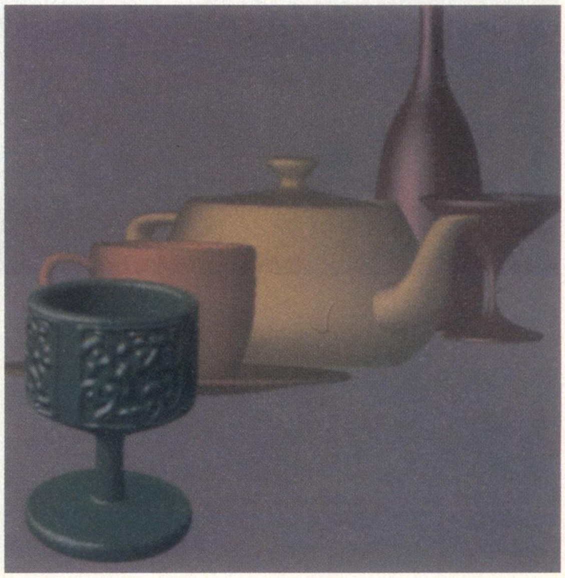

a


b

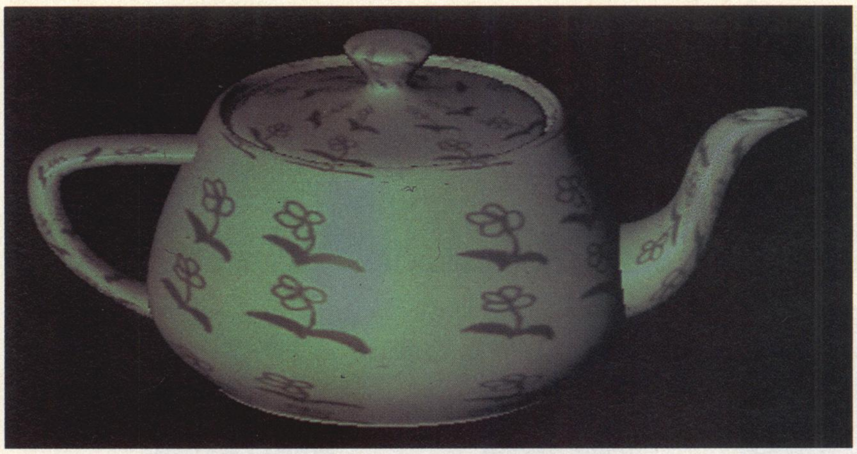
These two images, also owned by Omnibus, show (a) an interesting development of the teapot, and (b) the teapot in a complete table setting.



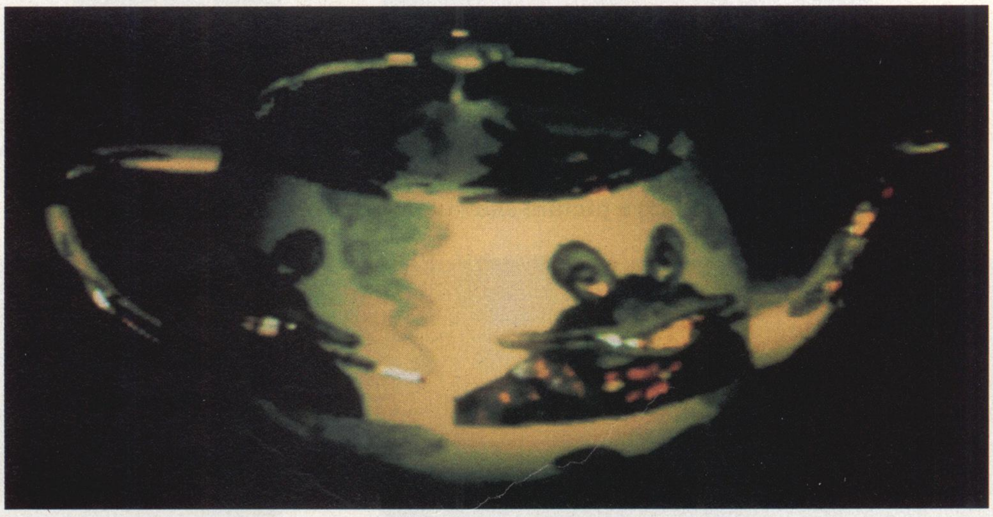
Shaded and texture mapped with specular reflectance by James F. Blinn while at the University of Utah in 1976-77.



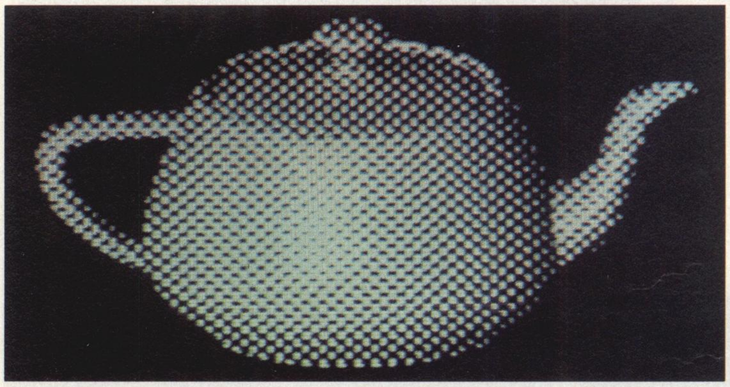
A composite of renderings by Turner Whitted (red bottle and glass), Loren Carpenter, Jeff Lane (teapot and cup), and James Blinn (green chalice) in 1977.



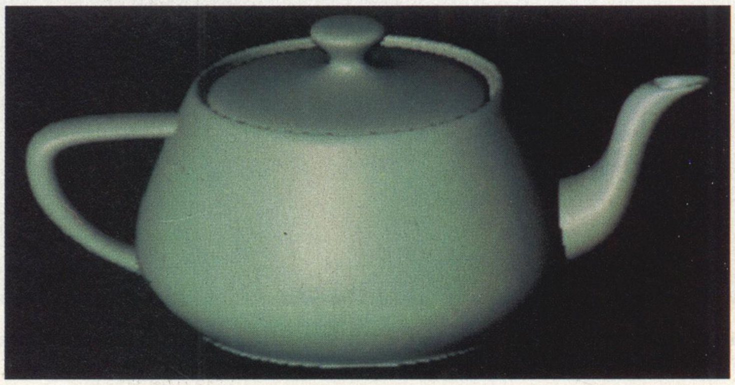
James Blinn's shaded teapot with painted texture map and specular reflectance from his 1976-77 Utah period.



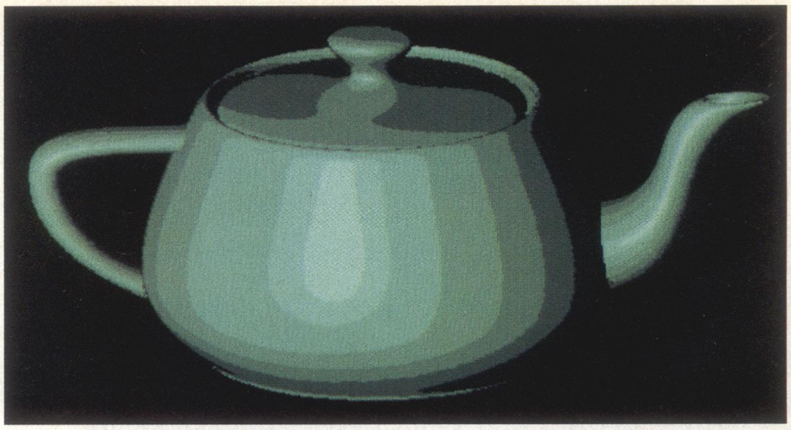
A teapot texture mapped with a painted picture by Lance Williams, done at NYIT, 1977.



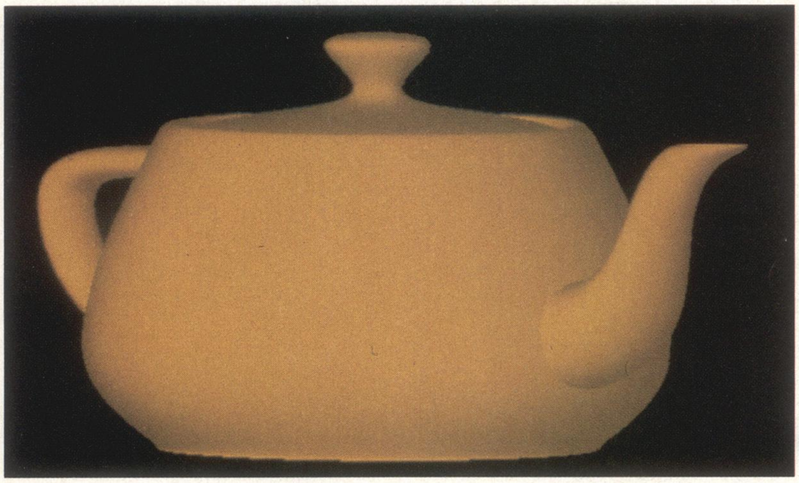
James Blinn's black/white shaded teapot with 1-bit halftone dithering.



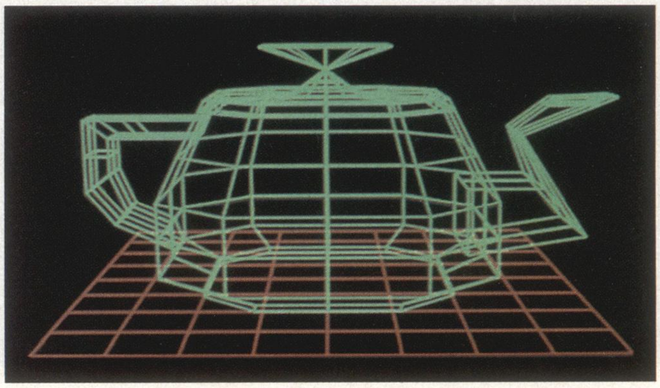
Teapot with specular reflectance (aliasing), by James Blinn.



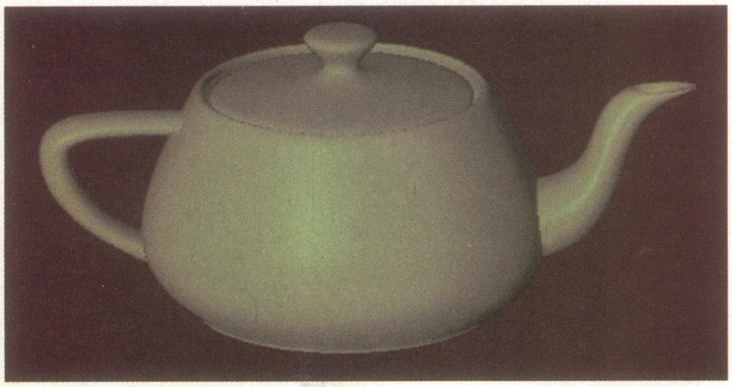
Shaded 3-bit teapot with threshholding, by James Blinn when he was at Utah during 1976-77.



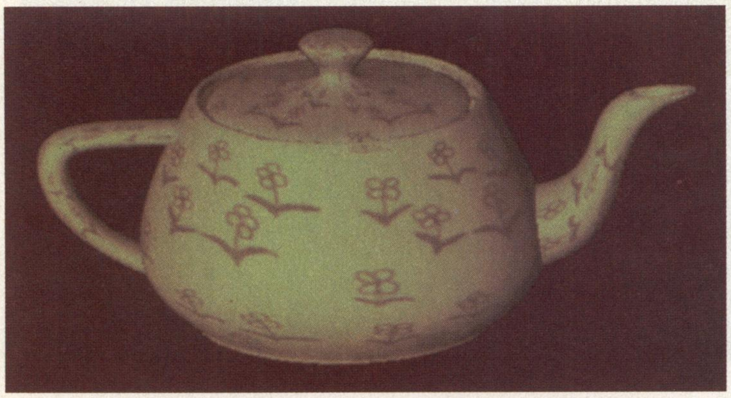
Teapot rendered by Loren Carpenter and Jeff Lane at Boeing, 1977.

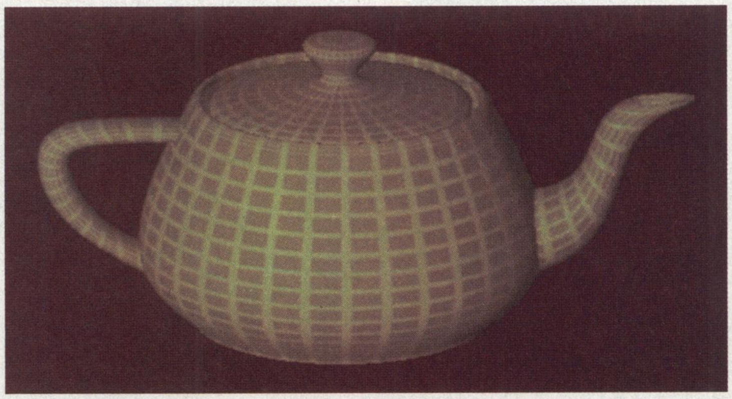


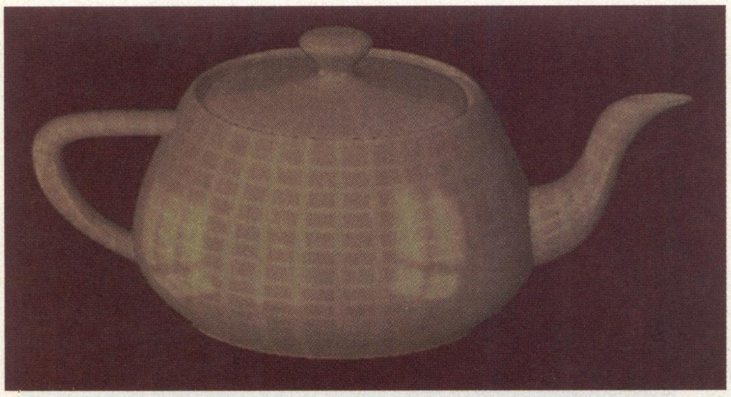
The following images are all by James Blinn. Map of control points for a teapot on a surface.

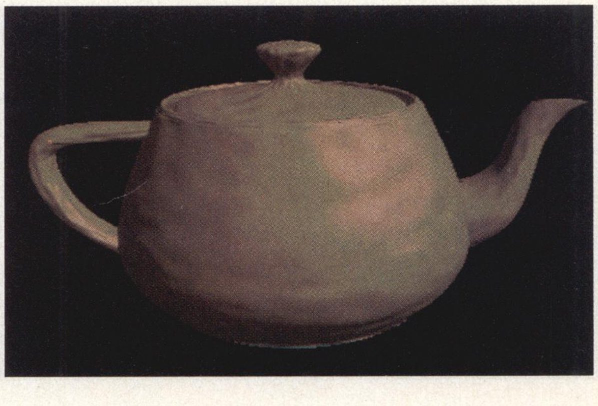


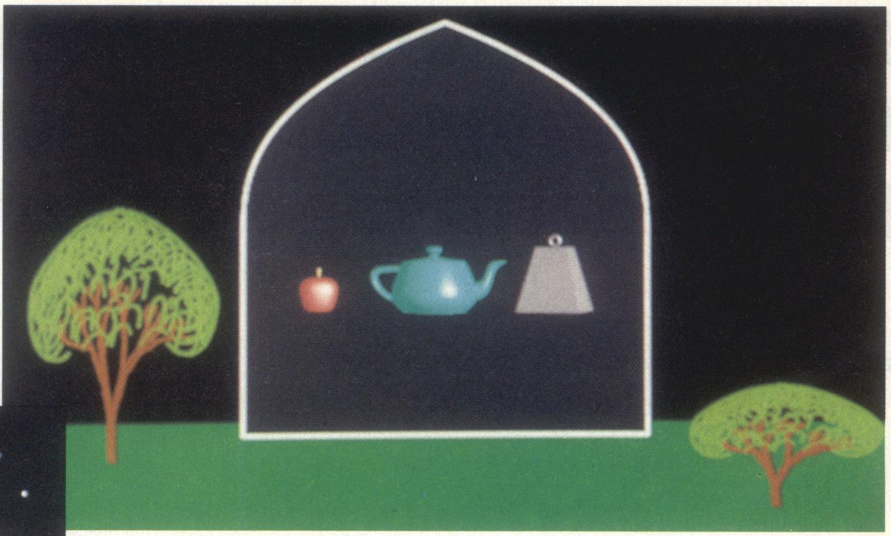
Shaded teapot antialiased.


A shaded teapot with painted texture map.


Shaded grid texture mapped on teapot.

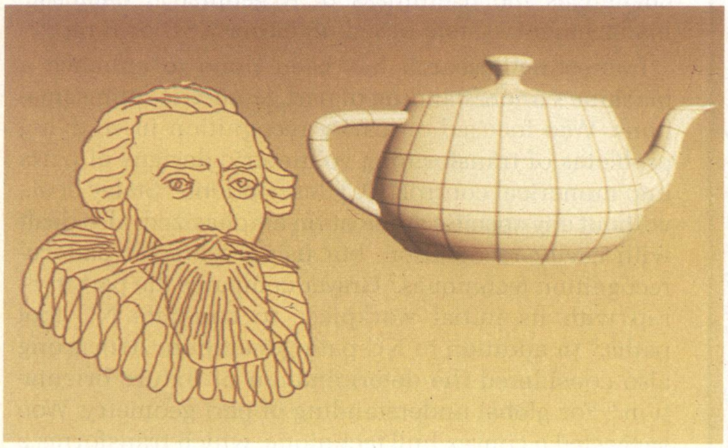

Shaded antialiased teapot with environment mapping (not ray traced), and grid mapped with specular reflectance.


Shaded and bump mapped.



Einstein thought experiment acceleration versus gravity (for *The Mechanical Universe* 1985), Computer Graphics Lab at the Jet Propulsion Laboratory.

Greeks finding the area of an irregular shape (*Mechanical Universe*, 1985) CGL, Jet Propulsion Laboratory.



Kepler and the teapot (for *Mechanical Universe*) CGL, Jet Propulsion Laboratory.