Misc3d—A Package to Draw 3D Graphs

Dai Feng (The University of Iowa), Luke Tierney (The University of Iowa)

Abstract

We introduce several functions from the misc3d package, including functions to compute and render isosurfaces (three dimensional contour plots) and functions to draw scenes consisting of one or more surfaces described by triangular mesh data structures. Rendering can be done using rgl, standard graphics, or grid graphics. We describe the algorithms used to compute isosurfaces—the original marching cubes algorithm and the marching cubes 33 algorithm which solves the face and internal ambiguity problems. We illustrate the power of misc3d's rendering functions by showing several views of the Utah teapot, a classic computer graphics example, and we present three illustrations of statistical applications.

The Marching Cubes Algorithm

The `marching cubes` (MC) algorithm, a well known high-resolution isosurface extraction method used in volume data visualization, was first studied by Lorensen and Cline (1987). MC produces the isosurface of $F(x, y, z) = \alpha$ by a divided and conquer method. The basic idea of MC is that the whole space can be divided into a set of cubes and decisions are made separately for each cube on how the intersection of the surface with the cube is represented by one or more triangles. MC was first proposed for and is widely used in medical image rendering. MC is also used in other fields, for example for displaying a volume of oil in a geological volume. In statistics it can be used to draw a three dimensional contour plot of a density function.

After dividing the space into cubes, each cube is examined. If one or more vertices (corners) of a cube have values less than the user-specified isovalue (negative), and one or more have values greater than this value (positive) then the cube must contribute some components of the isosurface. (Throughout the poster, vertices are considered to have the same values if they are either all greater or all smaller than the isovalue.) After determining which edges of the cube are intersected by the isosurface, a triangular topological representation of the surface can be constructed. Since there are 8 vertices for each cube and the value of each vertex can be either negative or positive, there are $2^8 = 256$ possible cases for each cube. Due to topological equivalence by rotation and switching between the positive and negative values, however, there are in total 15 configurations shown in Figure 1, which was generated based on lookup tables in `contour3d`. There is no triangle in configuration 0, since values of all vertices are either all positive or all negative. For configuration 1, all vertices, except the one at front left lower corner, have the same values. Therefore, the isosurface separates the unique vertex from the others. The topological representation of the isosurface within a cube is constructed by one or more triangles. The vertices of these triangles, the points at which the isosurface intersects the cube edges, are determined by bilinear interpolation. The isosurface representation is obtained by going through all the cubes

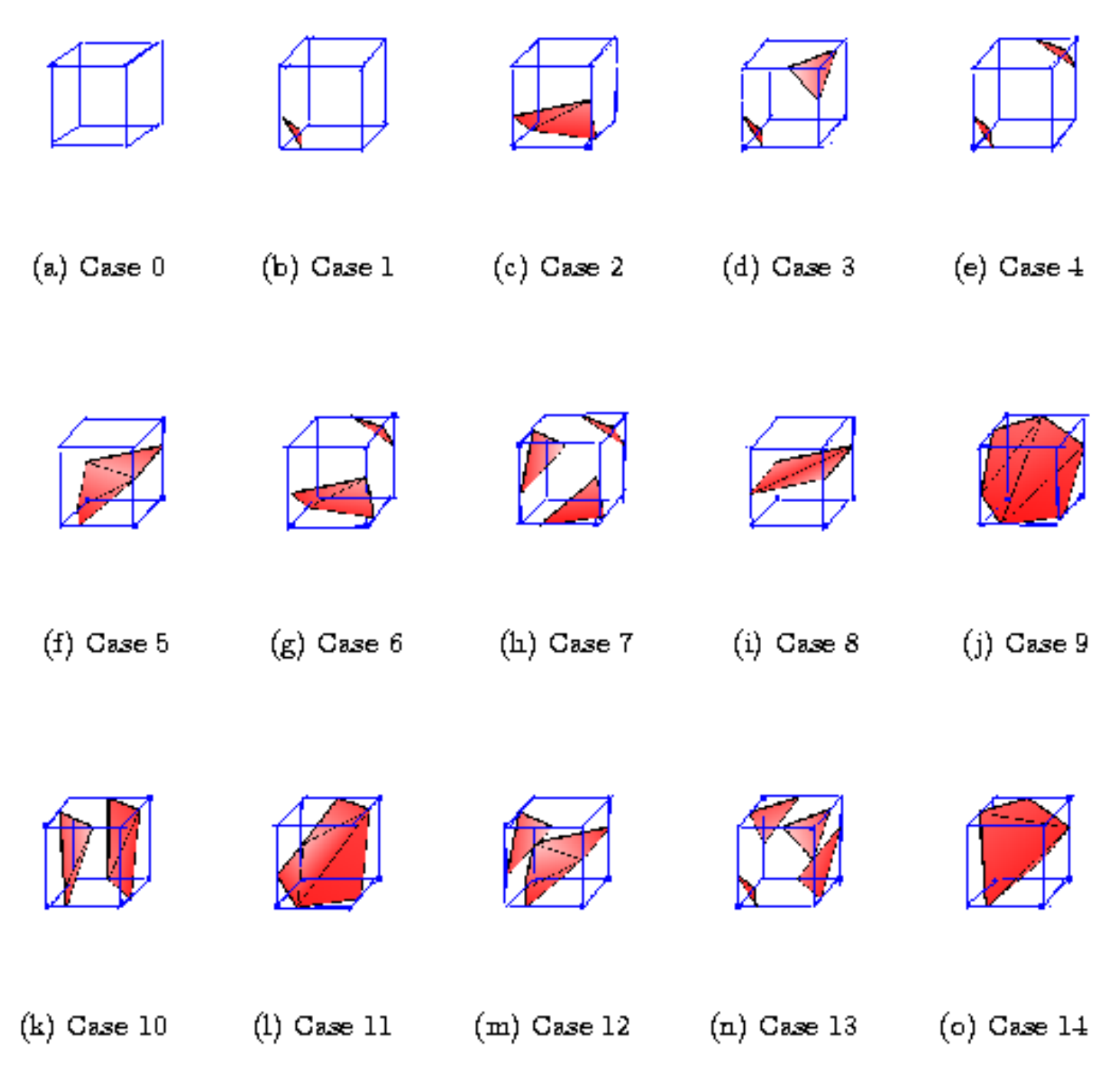

Figure 1. The Original Lookup Table of the Marching Cubes Algorithm.

Since its origin, a lot of research has been done to improve the quality of the topological representation produced by MC. Nielson and Hamann (1991) pointed out that there could be an ambiguity in the face of a cube when all four edges of the face are intersected. Face ambiguity arises when the vertices on diagonal corners have the same values, the values of vertices joined by edges differ. Face ambiguity is illustrated in Figure 2. If vertices A, C are positive and B, D are negative, then the isosurface can intersect the plane in two possible ways.

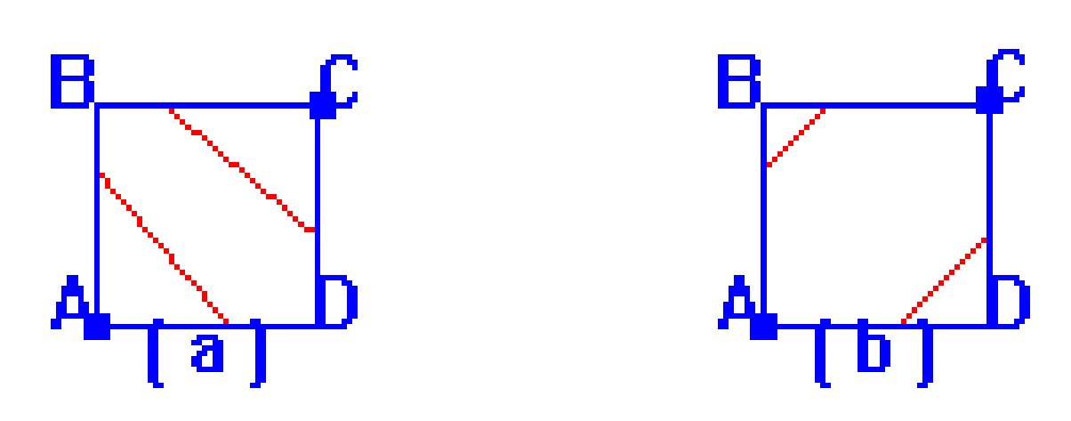

Figure 2. Illustration of face ambiguity

In addition to face ambiguities, Chernyaev (1995) recognized that, there are internal ambiguities, in terms of the representation of the trilinear interpolant in the interior of the cube. For example, in Figure 3, case 4 has two sub-cases. Two marked vertices could either be separated (case 4.1.1) or connected inside the cube (case 4.1.2). Furthermore, in order to make the triangular representation of the isosurface more conformable to the truth, an additional vertex might be needed (see case 7.3 in Figure 3 for example). The `contour3d` function mainly uses the algorithm suggested in Chernyaev (1995). The enlarged lookup table in Chernyaev 1995 is shown in Figure 3 and is derived from the lookup tables in `contour3d`. Note that there are still some sub-subcases which are not exhibited in the enlarged table. Take case 13.2 for example, there are 6 sub-subcases, and only the one with positive vertices on the top face being connected is shown.

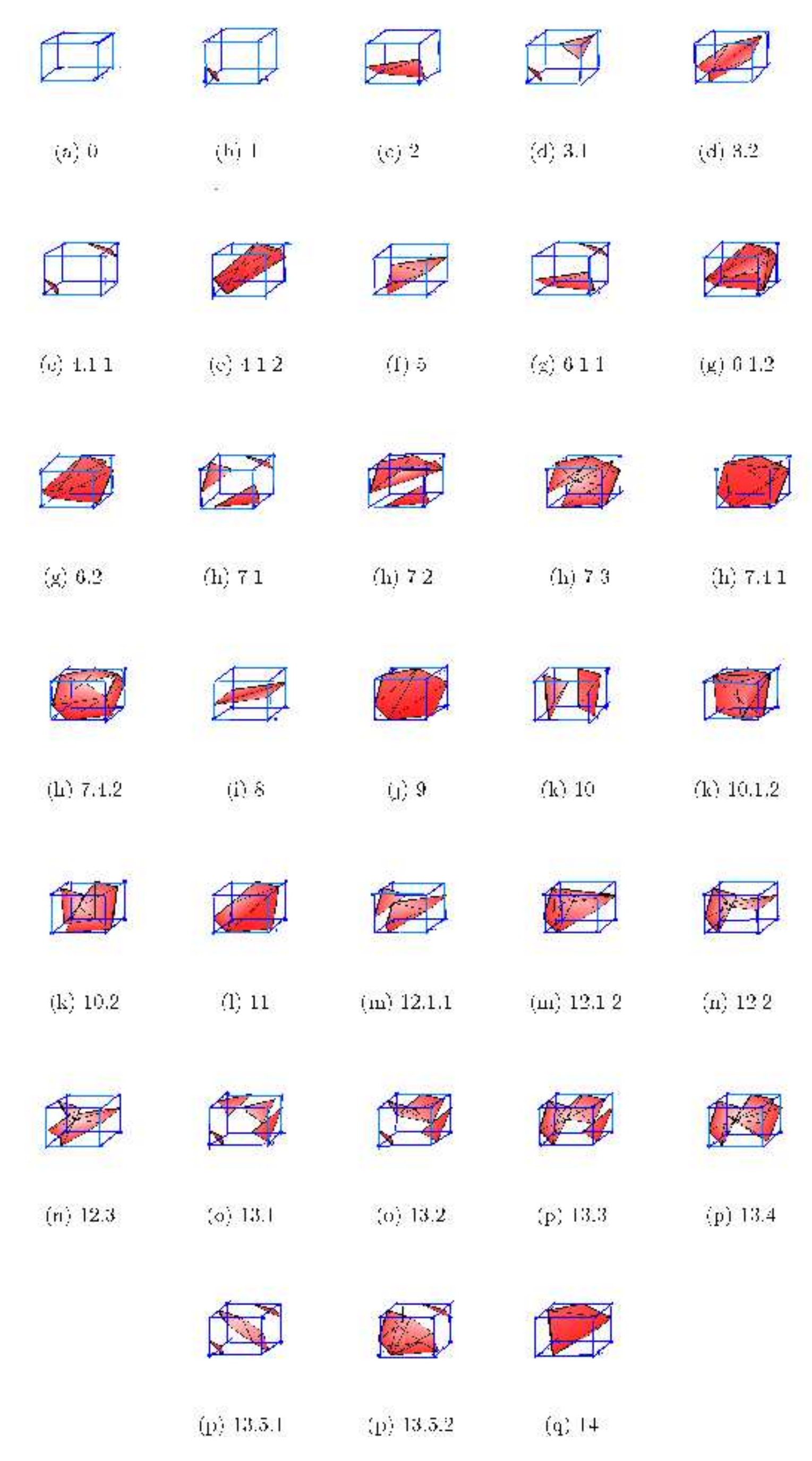

Figure 3. The Lookup Table of the Marching Cubes 33 Algorithm.

Illustrations of the Rendering Functions

We illustrate the power of misc3d's rendering functions by showing several views of the Utah teapot, a classic computer graphics example.

Different Rendering Engines

Rendering can be done using rgl, standard graphics, or grid graphics. Figure 4 shows the teapot rendered by three engines.

```
misc3d>  data(teapot)
misc3d>
misc3d>  haveRGL <- suppressWarnings(require(rgl,quietly=TRUE))
misc3d>
misc3d>  ttri <- makeTriangles(teapot$vertices, teapot$edges,
+                        color = "red", color2 = "green")
misc3d>  edges <- teapot$edges
misc3d>
misc3d>  ttriDull <- updateTriangles(ttri,material="dull")
misc3d>  ttriShiny <- updateTriangles(ttri,material="shiny")
misc3d>  ttriMetal <- updateTriangles(ttri,material="metal")
misc3d>  drawScene(ttri,screen=list(y=-30,x=40), scale = FALSE)
misc3d>  drawScene(ttri,screen=list(y=-30,x=40), scale = FALSE, engine="grid")
misc3d>  drawScene.rgl(ttri, color="green")
```

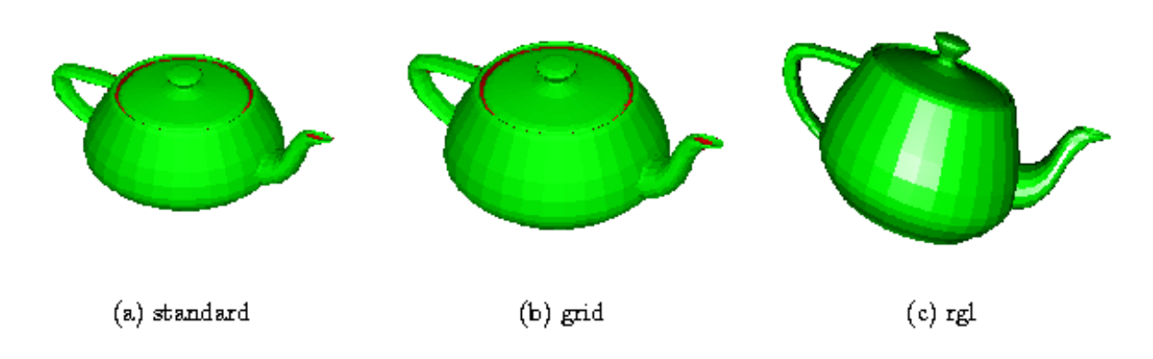

Figure 4. Different Rendering Engines.

Different Rendering Colors

Different colors could be used, as ilustrated in Figure 5.

```
misc3d>  drawScene(updateTriangles(ttriShiny,color2=grey.colors(ncol(edges))),
+            screen=list(y=-30,x=40), scale = FALSE)
misc3d>  drawScene(updateTriangles(ttriMetal, color2=heat.colors(ncol(edges))),
+            screen=list(y=-30,x=40), scale = FALSE)
```

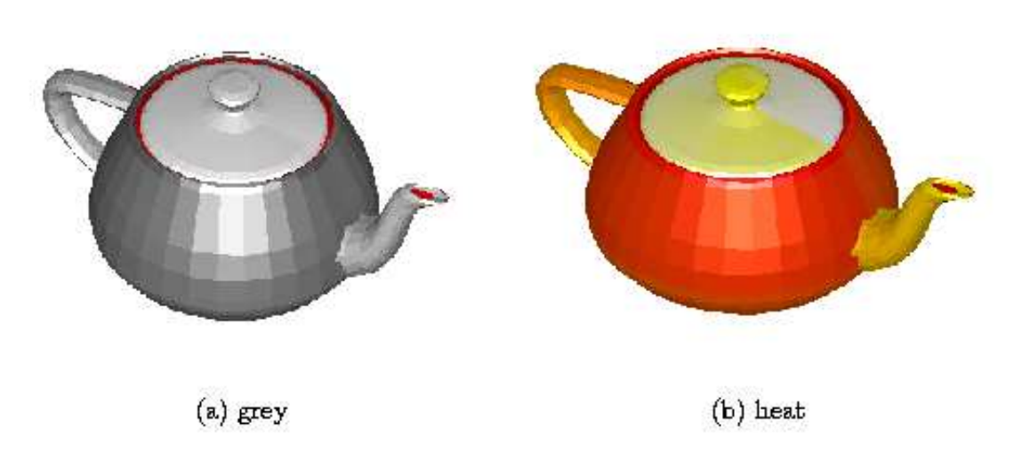

Figure 5. Different Rendering Colors.

Rendering Multiple Objects

Several objects could be drawn on the same graph by creating a list of appropriately positioned triangle mesh objects.

```
misc3d>  hc <- heat.colors(ncol(edges))
misc3d>  drawScene(list(updateTriangles(ttri, color2 = hc),
+                  translateTriangles(ttri,z=4)),
+             screen=list(y=-30,x=40), scale = FALSE)
```

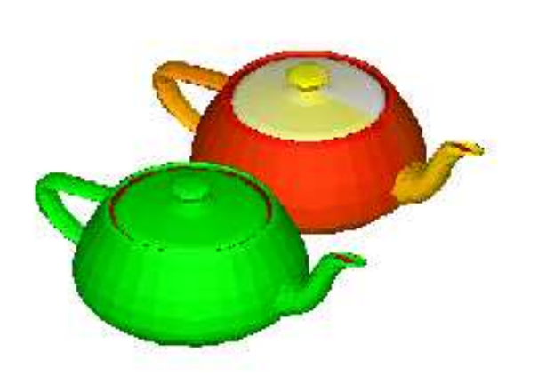

Figure 6. Multiple Teapots.

Nested Objects

Nested objects can be rendered using wire frame effects or using different transparency levels if supported by the rendering engine.

```
misc3d>  drawScene(list(updateTriangles(ttri,color="blue", fill=FALSE,
+             col.mesh="blue"),
+             scaleTriangles(updateTriangles(ttriMetal, color2="red"), 0.6)),
+             screen=list(y=-30,x=20,y=-140), scale = FALSE)
misc3d>  drawScene.rgl(list(updateTriangles(ttri, alpha = 0.5, color="blue"),
+                       scaleTriangles(ttriMetal, 0.6)))
```

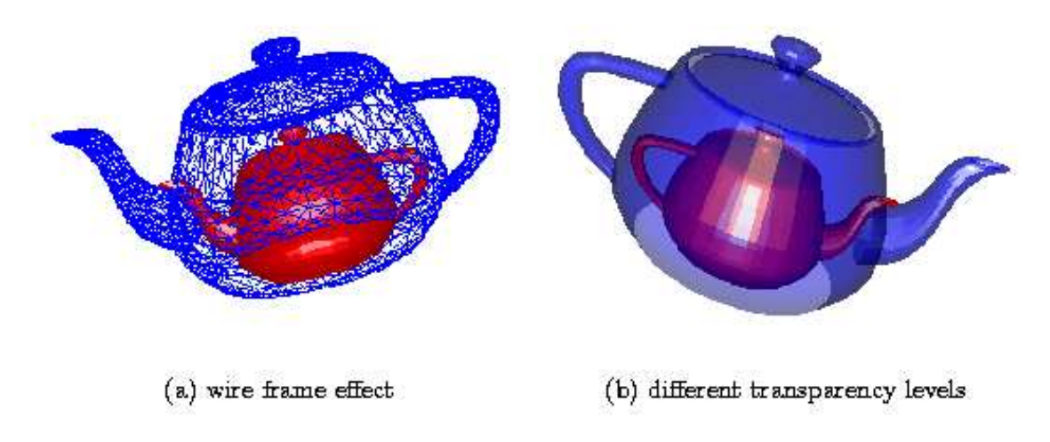

Figure 7. Nested Teapots.

Lighting and Shading

Rendering in the standard or grid engines is enhanced by using a lighting model. Shading can further enhance the results at the expense of higher computational cost. Two lighting models are supported by the functions phongLighting and perspLighting. The Phong lighting model adjusts colors based on view direction, light direction, and material properties. It incorporates ambient and diffuse light, which are the same color as the object, and specular light, which is a convex combination of the object color and the (white) light color. This is based roughly on the description in Foley et al. (1996). perspLighting implements approximately the same lighting model as the persp function. Figure 8 shows the effects of phongLighting with different materials, perspLighting, and the default.

```
misc3d>  drawScene(ttri,screen=list(y=-30,x=40), scale = FALSE)
misc3d>  drawScene(ttriDull,screen=list(y=-30,x=40), scale = FALSE)
misc3d>  drawScene(ttriShiny,screen=list(y=-30,x=40), scale = FALSE)
misc3d>  drawScene(ttriMetal,screen=list(y=-30,x=40), scale = FALSE)
misc3d>  drawScene(ttri,screen=list(y=-30,x=40),lighting=perspLighting,
+           scale = FALSE)
```

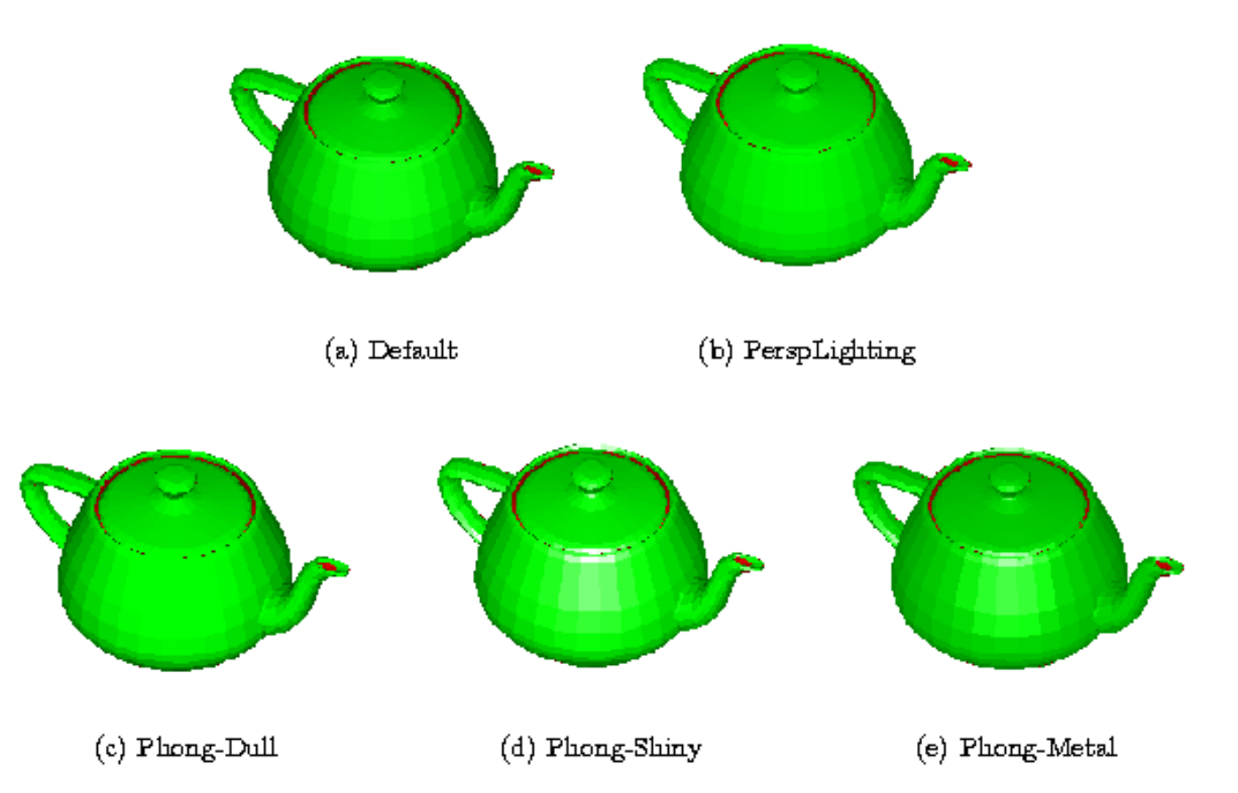

Figure 8. Different Lighting Functions.

The image can be further enhanced by subdividing the triangles one or more times and applying the Phong shading model to the sub-triangles. Figure 9 shows the effect of different Phong shading levels

```
misc3d>  drawScene(updateTriangles(ttriMetal, color2 = hc, smooth = 1),
+            screen=list(y=-30,x=40), scale = FALSE)
misc3d>  drawScene(updateTriangles(ttriMetal, color2 = hc, smooth = 2),
+            screen=list(y=-30,x=40), scale = FALSE)
misc3d>  drawScene(updateTriangles(ttriMetal, color2 = hc, smooth = 3),
+            screen=list(y=-30,x=40), scale = FALSE)
```

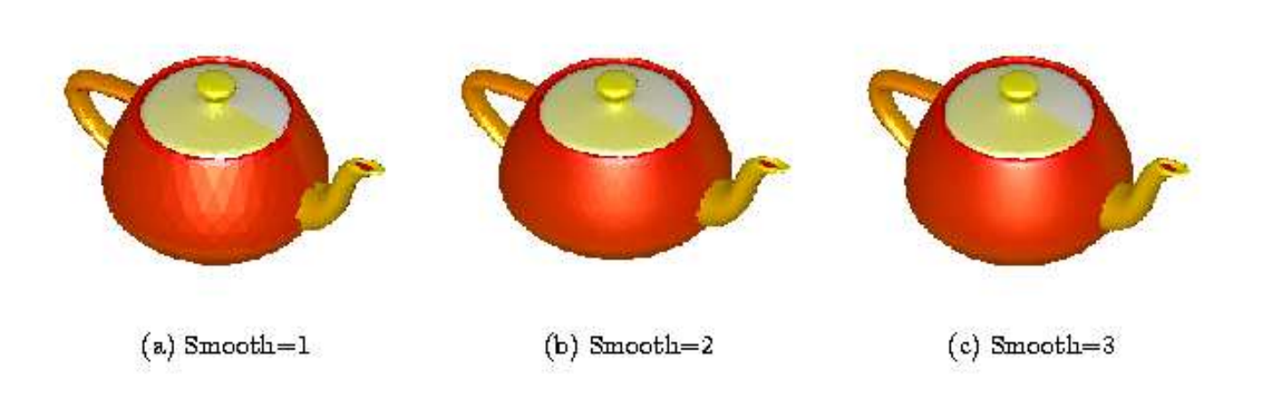

Figure 9. Different Phong Shading Levels.

Three Exampless of Statistical Applications

Contours of a Tri-Variate Density

Figure 10 shows nested contours of a mixture of three tri-variate normal densities. Case 1 uses transparency and case 2 uses a cut-out strategy to show the nested contours.

```
misc3d>  nmix3 <- function(x, y, z, m, s) {
misc3d+    0.4 * dnorm(x, m, s) * dnorm(y, m, s) * dnorm(z, m, s) +
misc3d+    0.3 * dnorm(x, -m, s) * dnorm(y, -m, s) * dnorm(z, -m, s) +
misc3d+    0.3 * dnorm(x, m, s) * dnorm(y, -1.5 * m, s) * dnorm(z, m, s)
misc3d+  }
misc3d>  f <- function(x,y,z) nmix3(x,y,z,.5,.5)
misc3d>  g <- function(n = 40, k = 5, alo = 0.1, ahi = 0.5,
misc3d+                cmap = heat.colors) {
misc3d+    th <- seq(0.05, 0.2, len = k)
misc3d+    col <- rev(cmap(length(th)))
misc3d+    al <- seq(alo, ahi, len = length(th))
misc3d+    x <- seq(-2, 2, len=n)
misc3d+    contour3d(f,th,x,x,x,color=col,alpha=al)
misc3d+    rgl.bg(col="white")
misc3d+  }
misc3d>  g(40,5)
misc3d>  gs <- function(n = 40, k = 5, cmap = heat.colors, ...) {
misc3d+    th <- seq(0.05, 0.2, len = k)
misc3d+    col <- rev(cmap(length(th)))
misc3d+    x <- seq(-2, 2, len=n)
misc3d+    m <- function(x,y,z) x > .25 | y < -.3
misc3d+    contour3d(f,th,x,x,x,color=col, mask = m, engine = "standard",
misc3d+              scale = FALSE, ...)
misc3d+    rgl.bg(col="white")
misc3d+  }
misc3d>  gs(40, 5, screen=list(z = 130, x = -80), color2 = "lightgray",
misc3d+    cmap=rainbow)
```

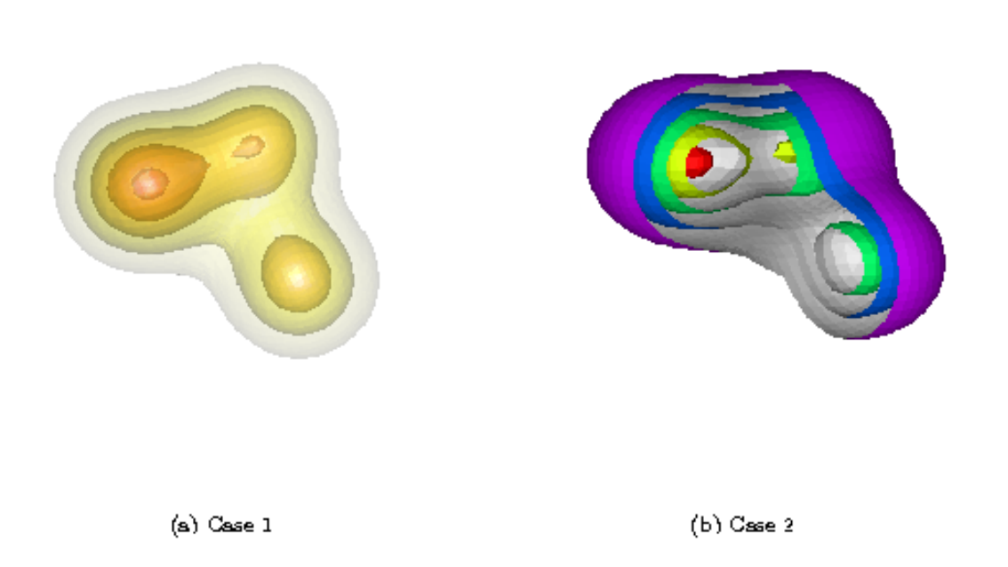

Figure 10. Nested Contours of Mixture of Three Tri-variate Normal Densities.

Figure 11 shows nested contours of a kernel density estimate for the iris data estimated using a 3D analog to kde2d in MASS.

```
misc3d>  library(MASS)
misc3d>  library(misc3d)
misc3d>  source("kde3d.R")
misc3d>  data(iris)
misc3d>  r <- kde3d(x=iris$Sepal.Length, y=iris$Sepal.Width, z=iris$Petal.Length)
misc3d>  rgl.points(x=iris$Sepal.Length,y=iris$Sepal.Width, z=iris$Petal.Length,
misc3d+              size=2, color="red")
misc3d>  axes3d()
misc3d>  rgl.bg(color="white")
misc3d>  contour3d(r$d, x=r$x, y=r$y, z=r$z,
misc3d+            lev=c(0.15, 0.05),
misc3d+            alpha=c(0.4,0.5),
misc3d+            color=c("red", "yellow"), add=T)
misc3d>  title3d(xlab="Sepal.Length",ylab="Sepal.Width",zlab="Petal.Length")
```

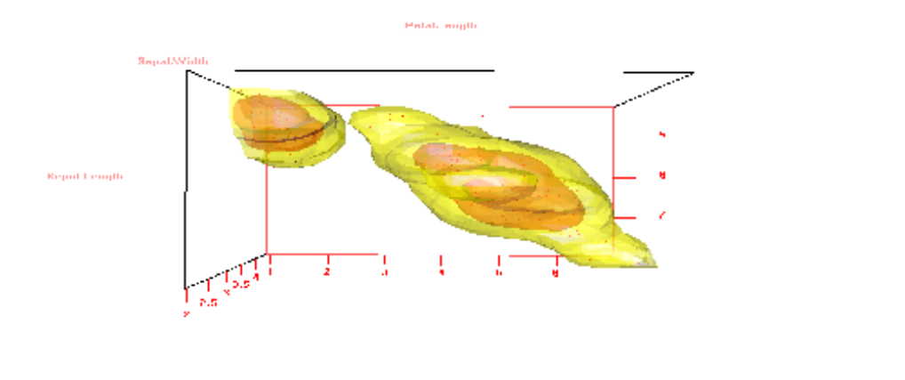

Figure 11. Nested Contours of a Kernel Density Estimate for the Iris Data.

Ratio-of-Uniforms Sampling Region

By the *Ratio-of-Uniforms* method, if V, U are uniform on

$$\mathcal{A} = \{(v, u) : v \in \mathbb{R}^d, 0 < u < \sqrt[d+1]{f(v/u + \eta)}\}$$

then $X = V/U + \eta$ has density proportional to f. Figure 12 shows $\mathcal{A}$ for the binormal distribution with $\mu = (-5, 5)$, $\sigma = (1, 1)$ and $\rho = 0.75$ based on $\eta = (0, 0)$ (green) and $\eta = (5, -5)$ (red).

```
misc3d>  parametric3d(fx=function(u, v) (u) * exp(-0.5 * ((u+5)^2 + (v-5)^2 -
misc3d+                 2 * 0.75 * (u+5) * (v-5))/sqrt(1-.75^2))^(1/3),
misc3d+                fy=function(u, v) (v) * exp(-0.5 * ((u+5)^2 + (v-5)^2 -
misc3d+                 2 * 0.75 * (u+5) * (v-5))/sqrt(1-.75^2))^(1/3),
misc3d+                fz=function(u, v) exp(-0.5 * ((u+5)^2 + (v-5)^2 -
misc3d+                 2 * 0.75 * (u+5) *(v-5))/sqrt(1-.75^2))^(1/3),
misc3d+                u = qcauchy((1:100)/101)-5, v = qcauchy((1:100)/101)+5,
misc3d+                color="green", box=TRUE)
misc3d>  parametric3d(fx = function(u, v) (u) * exp(-0.5 * (u^2 + v^2 -
misc3d+                 2 * 0.75 * u * v)/sqrt(1-.75^2))^(1/3),
misc3d+                fy = function(u, v) (v) * exp(-0.5 * (u^2 + v^2 -
misc3d+                 2 * 0.75 * u * v)/sqrt(1-.75^2))^(1/3),
misc3d+                fz = function(u, v) exp(-0.5 * (u^2 + v^2 -
misc3d+                 2 * 0.75 * u *v)/sqrt(1-.75^2))^(1/3),
misc3d+                u = qcauchy((1:100)/101), v = qcauchy((1:100)/101),
misc3d+                color="red", add=T, box=TRUE)
```

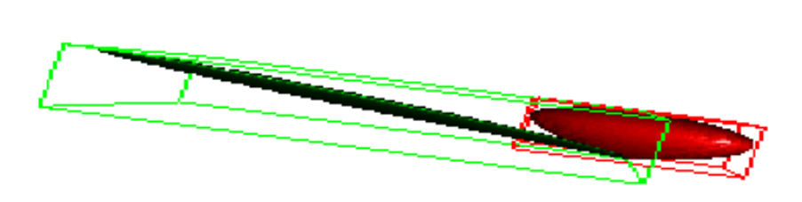

Figure 12. Ratio-of-Uniforms Sampling Region for Bivariate Normal Distribution.

Active Regions of the Human Brain

The left part of figure 13 shows the contour of a brain along with two contours of active regions of the human brain under stimulus, based on measured intensity differences between two tasks (one of them is the control) in a PET experiment. The three planes view of the active regions is shown on the right.

```
misc3d>  library(AnalyzeFMRI)
misc3d>  library(misc3d)

misc3d>  temp<-f.read.analyze.volume("template.img")
misc3d>  temp<-aperm(temp,c(1,3,2,4))
misc3d>  contour3d(temp[,,95:1,1],lev=10000,alpha=0.5)
misc3d>  tm<-f.read.analyze.volume("tmap1-8.img")
misc3d>  tm2 <- tm
misc3d>  tm[tm>=5] <- 5
misc3d>  contour3d(ifelse(temp[,,95:1,1] > 10000, tm[,,95:1], 0),
+               lev=5,color="yellow",add=TRUE)
misc3d>  tm[tm>=4] <- 4
misc3d>  contour3d(ifelse(temp[,,95:1,1] > 10000, tm2[,,95:1], 0),
+               lev=4,color="red",alpha=0.3,add=TRUE)
misc3d>  tm2[tm2 < 4] <- 0
misc3d>  tm2[tm2 >=4 & tm2 <5] <- 1
misc3d>  misc3d>  tm2[tm2 >=5] <- 2
misc3d>  slices3d(tm2,col=c("white", "red", "yellow"),main="Three Planes View")
```

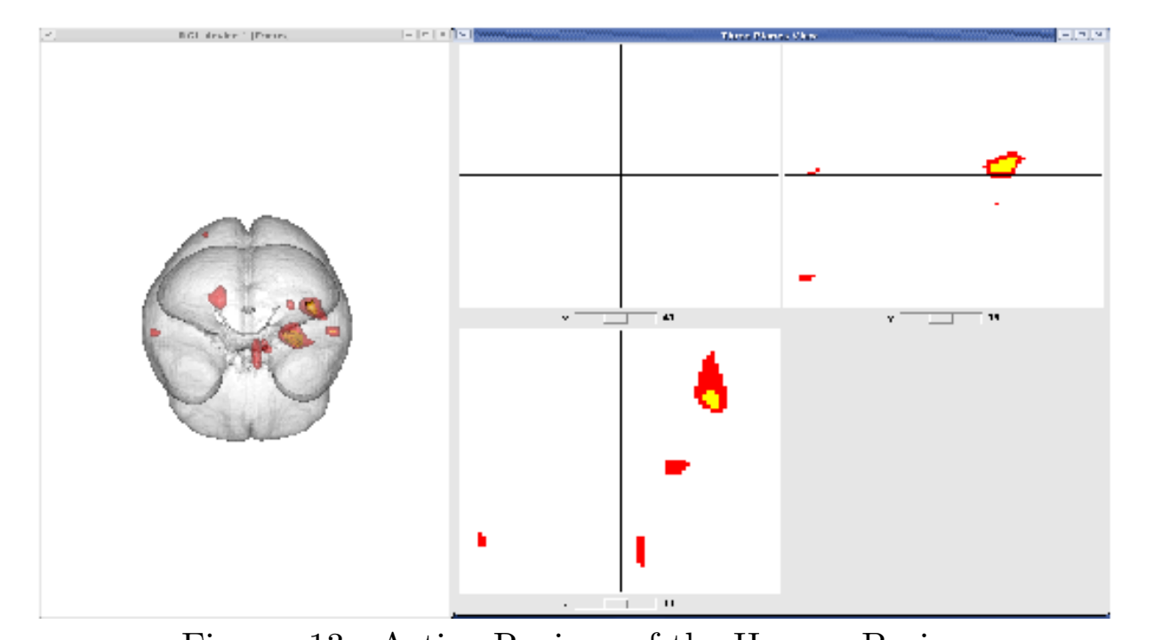

Figure 13. Active Regions of the Human Brain.

REFERENCES

Chernyaev, E. V. 1995. Technical Report CN/95-17, CERN, Institute for High Energy Physics.

Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. 1996. *Computer Graphics: Principles and Practice in C* (second ed.). Addison-Wesley Professional.

Lorensen, W. E., & Cline, H. E. 1987, Computer Graphics, *21*(4), 163.

Nielson, G. M., & Hamann, B. 1991. In *VIS '91: Proceedings of the 2nd conference on Visualization '91*, Los Alamitos, CA, USA, pp. 83–91. IEEE Computer Society Press.

This poster was prepared with Brian Wolven's Poster LaTeX macros v2.1.