Object File Format

HyperCuber 2.0 is the first version of HyperCuber to support user-defined objects. The object file format is still very rough, and will certainly improve in the future. For the moment, it works, barely, and that's something.

The format, in theory, supports three kinds of primitives: points, line segment paths, and polygons. An actuality, it supports only line segment paths. I wrote some code to draw the points and polygons, but I haven't tested it, and it would be a miracle if it worked. I guarantee, in fact, that polygons do not obscure each other correctly, since there is absolutely no hidden line removal code in HyperCuber 2.0. So unless you're daring or curious, stick to line segment paths.

A line segment path is just a list of points. When drawn, these points will be connected consecutively with line segments (all of the same color). So a square might be a list of five points: (0, 0), (1, 0), (1, 1), (0, 1), (0, 0). Note that HyperCuber does not automatically close the path; if you want a closed path you have to repeat the first point at the end.

HyperCuber is fairly fast, and part of this speed results from its policy of only projecting or rotating a point once. For instance, our square has two line segments connected to (0, 0), but HyperCuber only computes the projection or rotation of (0, 0) once, and uses the result to draw both line segments. HyperCuber does this by keeping a list of all the points in an object. Every time you refer to a point, you refer to it not by coordinates but by the index of that point into the vertices table. So if our vertices table were T[1] = (0, 0), T[2] = (1, 0), T[3] = (1, 1), T[4] = (0, 1), we could draw the square by connecting line segments in the order 1-2-3-4-1. HyperCuber looks up 1 in the table and sees that that's really (0, 0) and uses those coordinates to draw it. If we want to rotate the square, we can just rotate each point in the table and the use the same 1-2-3-4-1 to draw. Note in particular that we don't need to rotate point 1 twice, even though it's referred to twice in the description.

HyperCuber uses this same sort of table setup for the colors; all colors to be used are defined at the beginning of the file and are referred to by index.

Below is the file format. Basically, an object file is a list of vertices used, followed by a list of colors used, followed by a list of primitives. Note that HyperCuber does not yet support comments in object files; the comments below are there only to describe the different parts of the file. For a specific example of an object file, see the Sample Object File chapter.

P.S. If anyone makes some nifty objects, please mail them to me; I would love to see them, and I might even include them with the next release!

============================== File format
==============================

1

; version number

d

; dimensions of the object (i.e. 4 for a 4D object)

0

; reserved (must be 0)

0

; reserved (must be 0)

n

; number of vertices

(x1, x2, ..., xn) ; vertices (coordinates)

(x1, x2, ..., xn)

. . .

(x1, x2, ..., xn)

c

; number of colors

r1, g1, b1

; red/green/blue components of colors (each up to 16-bit)

r2, g2, b2

. . .

rc, gc, bc

m

; number of primitives

primitive 1

primitive 2

. . .

primitive m

[eof]

; end of file; don't actually type [eof]

========== Primitives format ==========

p

; primitive type

primitive description
;

========== Primitive Types and Descriptions ==========

Type 1: point
Description:

c

; color of point (index into color list)

p

; coordinates of point (index into vertex list)

Type 2: line segment path
Description:

c

; color of line segment (index into color list)

n

; number of points in path

p1

; points in path (indices into vertex list)

p2

...

pn

; (note: pn is NOT automatically connected to p1)

Type 3: filled polygon
Description:

ic

; color of interior of polygon (index into color list,
; transparent if 0)

bc

; color of boundary of polygon (index into color list,
; not drawn if 0)

n

; number of points in polygon

p1

; points in polygon (indices into vertex list)

p2

...

pn

; (note: pn is automatically connected to p1 to make
; polygon closed)