Human writable glyph manifold UTF-8 extension

Starting sequence: 1111 1011 0000 0010
Memory use: 284048 bits (including the starting sequence); or you could say 35506 bytes per character.
Camera: 104 bits
Lights: 1120 bits
Swatches: 20664 bits
Points: 262144 bits

(This system of making glyphs is theoretically human writable, not as in you can write in natural language and get what you want back out, no, this isn't ChatGPT. Rather, this system is "human writable" as in a human artist could take a ruler and measure out something the artist drew, and figure out the coordinate conversion into a cube with corners at (-100, -100, -100), (100, -100, -100), (-100, 100, -100), (-100, -100, 100), (100, 100, 100), and then type in binary the appropriate positions where points should show up in the model. This is "human writable" to the extent that my previous attempt at incorporating Python3 code in a Mandelbrot-set/Turing-Halting-Problem-method-of-image-compression "image glyph" UTF-8 extension was a bad idea because no human could possibly make the images I was thinking of in that system.)

Note: all "positions" and "coordinates" stated herein are made using 16 bit signed integers, where -32768 is linearly interpolated to the -100 position of a face of the bounding cube that everything inside the glyph exists within; 32767 is linearly interpolated to the 100 position of a face of the bounding cube that everything inside the glyph exists within. However, within a given point datablock (note: the word "point" here is referring to a very specific (x, y, z) coordinate of 1024 "points" whose coordinate is within that 200 side length cube), there is an "r" value such that that point can be considered as surrounded by a virtual "area of effect box" with a side length of 2*"r"; then later in that "point" datablock there are four listed "corners" using 12 bit signed integers. A value of -2048 for the x value of a "corner" puts that corner on the left face of that "area of effect box". A value of 2047 for the x value of a "corner" puts that corner on the right face of that "area of effect box; some value between -2048 and 2047 is linearly interpolated for where to put the corner between those two faces of the "area of effect box"; this is similarly applied for the y axis (-2048 at the face of the cube looking towards the bottom of the page with respect to the human reading the character later; 2047 at the face of the cube looking towards the top of the page with respect to the human reader (when the camera position is the bit sequence 0000 0000 0000 0000   0000 0000 0000 0000   0111 1111 (otherwise known as the "top-middle" of the bounding box of the effect) and the camera has the maximally far away bit string at 0000 0000 0000 0000   0000 0000 0000 0000   0000 0000 0000 0000
Note: to avoid confusion, there is no extra rotation parameter on the camera, so the writer of one of these glyphs can only use pitch and yaw, never roll (to use aircraft pilot terminology).

48 bits		Choose camera position: (x, y, z)
48 bits		Choose position maximally far away (at the plane that defines what the camera can't see because of something being "too far away") that will be in the center of the camera's field of view: (x, y, z)
1 bit:		if 0, use orthographic projection; if 1, then use perspective projection
7 bits:		an unsigned integer that is either interpreted as the size of the camera screen for orthographic projection; or, if using perspective projection, this number is interpreted as the number of degrees in the field of view (possibly simulating a wide angle lens).

Lights, maximum of 7.
2 bits: 00 if off; 01 if this is a “Sun” (directional light); 10 if Spotlight; 11 if Omnilight.
8 bits: red
8 bits: green
8 bits: blue
6 bits: “sharpness of shadows”, where 63 is maximum sharpness, 1 is diffuse but visible effect, 0 is no shadows.
48 bits: Initial “Point A” position: (x, y, z)
48 bits: “Point B” position: (x, y, z) (this is where Spotlight points to, and this defines the arrow of direction that the Sun looks towards.
32 bit float: R (maximum range, where the light doesn’t affect anything this far or farther from point A). This is a 32 bit float to make sure it operates consistently and allows for major expansions in the simulation later.

Swatches, 63 of them, with swatch 00 0000 being used at a point that doesn’t intend to be drawn.
96 bits Albedo
8 bits Alpha
32 bits Metallic
32 bits Specular
32 bits Roughness
96 bits Emission (Red, Green, Blue)
24 bits Transparency color (red, green, blue) (8 bits per color channel, since this isn’t defining a particular color, this is just establishing the hue angle that is preferred to the extent that photons that might have been allowed through by low alpha get blocked according to a probability distribution that reduces those photons by 1/(selectivity) for every 5 degrees of hue that color is away from the transparency color)
8 bits: Transparency selectivity (8 bits unsigned; 0 means no light gets through, even if alpha is low)

Points, 1024 of them: (each takes 256 bits)
6 bits: swatch color
48 bits: (x, y, z)
10 bits: “r” the unsigned integer quantity such that 1024 would have the internal bounding box of this “point” (as in, the bounding box that the later mentioned tetrahedron fits inside) as double the bounding box that the stuff of the glyph overall is supposed to fit inside.
12 bits: signed integer x coordinate of corner 1 of the tetrahedron (with -2048 at the left side of that inner cube; 2047 at the right side of that inner cube, or otherwise linearly interpolated)
12 bits: signed integer y coordinate of corner 1
12 bits: signed integer z coordinate of corner 1
36 bits: signed integers (x, y, z) of corner 2 of the tetrahedron
36 bits: (x, y, z) of corner 3 of tetrahedron
36 bits: (x, y, z) of corner 4 of tetrahedron
4 bits: signed integer x velocity of corner 1 in the animation (-8 means corner 1 hits the left side of the inner bounding box by the end of the animation; 7 means corner 1 hits the right side of the inner bounding box by the end of the animation)
4 bits: y velocity of corner 1 in the animation
4 bits: z velocity of corner 1 in the animation
12 bits: (x, y, z) velocity of corner 2 in the animation
12 bits: (x, y, z) velocity of corner 3 in the animation

12 bits: (x, y, z) velocity of corner 4 in the animation