AVA File Format (Version 1)
Document Updated 7/17/97

(1) Overall structure of the AVA file

Header
Vertex List
Patch List
4 Bytes (internal use)
Bones List
End Header

AVA file is binary.
Little Endian byte order.

(2) Header

The structure of the Header

“B” (1 byte)
ID (4 bytes) (internal use)
Version (4 bytes)

(3) Vertex List

The structure of the Vertex List

number of vertexes (4 bytes, int)
x coordinate 1 (4 bytes, float)
y coordinate 1 (4 bytes, float)
z coordinate 1 (4 bytes, float)
.....
x coordinate n (4 bytes, float)
y coordinate n (4 bytes, float)
z coordinate n (4 bytes, float)

IEEE floats.

(4) Patch List

A patch's control grid is laid out as below:

p1	p2	p3	p4
p5	p6	p7	p8
p9	p10	p11	p12
p13	p14	p15	p16

Since p1, p4, p16, and p13 define the quadrangle formed by the patch, and since the Vertex list contains these points, only their index is in the patch list.

p2, p3, p5, p8, p9, p12, p14, and p15 define the border of the patch as a convex hull.

p6, p7, p10, and p11 are the contrived inner points that define the NURB.

The structure of the Patch List

number of patches (4 bytes, int)
p1 index (4 bytes, int)
p4 index (4 bytes, int)
p16 index (4 bytes, int)
p13 index (4 bytes, int)
p2.x (4 bytes, float)
p2.y (4 bytes, float)
p2.z (4 bytes, float)
.....
p15.x (4 bytes, float)
p15.y (4 bytes, float)
p15.z (4 bytes, float)
.blue color (4 bytes, float)
green color (4 bytes)
red color (4 bytes)
ambiance (4 bytes, float)
roughness (4 bytes, float)
roughness scale (4 bytes, float)
reflectivity (4 bytes, float)
blue transparency (4 bytes, float)
green transparency (4 bytes, float)
red transparency (4 bytes, float)
refraction (4 bytes, float)
blue specular color (4 bytes, float)
green specular color (4 bytes, float)
red specular color (4 bytes, float)
specular size (4 bytes, float)
specular intensity (4 bytes, float)
width (4 bytes, int)
height (4 bytes, int)
optional size (4 bytes, int)

optional RLE data (use size)
.....

After the attribute information, an RLE compressed, 32-bit texture map is saved (if one exists). No texture map exists if the “width” and “height” values are 0. Otherwise, a “size” value indicates how large a chunk to read which contains the information.

Each color component scan line is packed separately, and packing never extends beyond a scan line.
Each color component is then saved out by row, (RGBA) ordered.

For example:
All of the red components of the first row are compressed together and saved out, followed by
All of the green components of the first row compressed together and saved out, followed by
All of the blue components of the first row compressed together and saved out, followed by
All of the alpha components of the first row compressed together and saved out, followed by

All of the red components of the second row are compressed together and saved out, followed by
All of the green components of the second row compressed together and saved out, followed by
All of the blue components of the second row compressed together and saved out, followed by
All of the alpha components of the second row compressed together and saved out
…
…

compression scheme:

For a given control byte 'n':
0 <= n <= 127 : use the next n + 1 bytes literally (no repetition).
-127 <= n <= -1 : use the next byte -n + 1 times.
n = -128 : no operation, not used.

See Unpacker.cpp for a code snip on loading and uncompressing the texture map.

(5) Bone List

The structure of the Bone list.

progeny flag (4 bytes, 0 or 1)
sibling flag (4 bytes, 0 or 1)
Name of Bone (n+1 bytes) NULL terminated
x Bone Translate (4 bytes, float)
y Bone Translate (4 bytes, float)
z Bone Translate (4 bytes, float)
x Bone Rotate (4 bytes, float)
y Bone Rotate (4 bytes, float)
z Bone Rotate (4 bytes, float)
x Bone Roll Handle (4 bytes, float)
y Bone Roll Handle (4 bytes, float)
z Bone Roll Handle (4 bytes, float)
number of Vertexs (4 bytes, int)
index 1 (4 bytes)
index 2 (4 bytes)
.....
index n (4 bytes)

6 End Header

End ID (4 bytes) (internal use)