MMD:Polygon Model Data

Hey~! This page is informative~!

(^_^) The content of this page may be of interest to readers, contributors and visitors.

This article in a nutshell states:

The **Polygon Model Data** (PMD) file format is the file format used to store 3D model information for the models used in the MikuMikuDance (Polygon Movie Maker) animation program.

As with any 3D modeling format, this is an extremely detailed file format which has no known documentation other than what many people around the Internet have reverse-engineered. The format is not based on any other known file formats, and so far is completely proprietary to MikuMikuDance.

Article originally written by alcexhim. I am currently working with him, and I edited this to make sense from what I have learned from making the new AniMiku rendering engine.

-Re:VB-P

In the left column, I have included the byte length of the entry, and the variable type that is used in native code (C or C++). In the right column, I have included either what the value of the entry should be, or what it represents if it is user-modifyable.

Contents [show]

File header Edit

File starts out with simple three char tag followed by a file version.

Data type	Value
byte[1] (char)	'P'
byte[1] (char)	'm'
byte[1] (char)	'd'
byte[4] (float)	'1.0'

Model header Edit

This section stores basic user-generated information about the model that follows the file header. This is only the Japanese model information. The English model information is stored in another section.

Data type	Description
Char[20]	The name of the character model, in Shift-JIS encoding
Char[256]	Comment for the model. This is used to provide information such as the model author, copyright information, etc.

Vertex information Edit

This section stores information about the vertices in the model. It starts out with a byte[4] (unsigned long in C++) describing the number of vertices in the model.

The following setup will occur once for each vertex.

Data type	Description
byte[4] (float)	X-coordinate of the vertex's position.
byte[4] (float)	Y-coordinate of the vertex position.
byte[4] (float)	Z-coordinate of the vertex position.
byte[4] (float)	X-coordinate of the vertex normal.

byte[4] (float)	Y-coordinate of the vertex normal.
byte[4] (float)	Z-coordinate of the vertex normal.
byte[4] (float)	U-coordinate of the vertex texture.
byte[4] (float)	V-coordinate of the vertex texture.
byte[2] (unsigned short)	Bone 0 id (index in the bone list. First bone to effect the position of this vertex in vertex/geometry skinning)
byte[2] (unsigned short)	Bone 1 id (index in the bone list. Second bone to effect the position of this verted in vertex/geometry skinning)
byte[1] (char)	Bone 0 integer weight from 0 to 100 (Weight of the first bone for vertex/geometry skinning. Weight of second bone calculated by 100-bone0Weight). As you can see, normalizing this weight requires you to divide by 100 and not by 255 or other...
byte[1] (char)	Edge Flag (Determines if this vertex should be used to draw the edge line around the model)

Index information Edit

This section describes the indices of the model. The section starts out with a byte[4] (unsigned long in C++) describing how many indices are in the model (MUST be a multiple of 3). These are used for connecting the vertices together and drawing the triangles that are used to form the 3D object.

Here is a quick explanation: The indices are stored in pairs of 3. Each of these 3 indices represents a corner of a triangle, and points to a vertex in the vertex list. In most 3D APIs (like Direct3D or OpenGL) the API handles drawing these for you with draw functions called "Indexed Drawing" functions. With these, the API will automatically look into the list of indices (called an "Index Buffer") and use each group of three to build a triangle using the index into the list of vertices (called "Vertex Buffer") for each corner of the triangle. This eliminates the need for duplicated vertices, thus saving precious processing power and video memory.

For each index:

Data type	Description
byte[2] (unsigned short)	One vertex ID in the list of vertices. These are arranged in groups of 3.

Materials information Edit

This section stores information about the materials in the model. All RGBA values are on a scale of 0.0-1.0 which is what most 3D APIs use, not 0-255. It starts out with a byte[4] (unsigned long in C++) describing the number of materials in the model. Then, for each material in the model:

Data type	Description
byte[4] (float)	R-value of the material's diffuse color.
byte[4] (float)	G-value of the material's diffuse color.
byte[4] (float)	B-value of the material's diffuse color.
byte[4] (float)	A-value (alpha) of the material's diffuse color.
byte[4] (float)	Specularity (Shininess) of the material.
byte[4] (float)	R-value of the material's specular color.
byte[4] (float)	G-value of the material's specular color.
byte[4] (float)	B-value of the material's specular color.
byte[4] (float)	R-value of the material's ambient color.
byte[4] (float)	G-value of the material's ambient color.
byte[4] (float)	B-value of the material's ambient color.
byte[1] (unsigned char)	Toon number. Value of 255 means toon 0 (no toon texture). Otherwise it is a value of 0-9 reprsenting an index into the array of toons (MMD uses 10 total toon textures)
byte[1] (unsigned char)	Edge Flag (determines if the edge line should be drawn around this material)
byte[4] (unsigned long)	Number of indices into the vertex list that are effected by this material. (Divide by 3 to get the number of triangle surfaces that are colored by this material)
char[20]	The file name corresponding to the texture applied to this material. May be left blank. If material has a sphere map, the texture file and sphere map file are separated by a "*". Example: "tex0.bmp*sphere01.spa"

Bones information Edit

This section stores information about the bones in the model. It starts out with a byte[2] (unsigned short in C++) describing the number of bones in the model. Then, for each bone in the model:

Data type	Description

char[20]	The name of the bone in Japanese (shift-jis).
byte[2] (short)	The index into the bone list which represents the parent of this bone. Will be -1 if there is no parent.
byte[2] (short)	The index into the bone list which represents the child of this bone. Will be -1 if there is no child.
byte[1] (char)	Bone Type. Look in PMD Editor to see the different bone types and what number represents each type.
byte[2] (short)	"Target Bone" This can be 1 of 2 things. See below.
byte[4] (float)	X-coordinate of the bone's position
byte[4] (float)	Y-coordinate of the bone's position
byte[4] (float)	Z-coordinate of the bone's position

Target Bone Information:

- If the bone is an IK follow bone (type 4):
  - This is the bone index of the IK bone that this bone is effected by.
- If the bone is a Co-Rotate bone (type 9, more information can be found by searching for MMD bone types)
  - This is the "co-rotate coefficient", or the value used to calculate the rotation for this bone.

Inverse Kinematics information Edit

This section stores information about the IK in the model. Inverse Kinematics is a way of animating a group of related bones all at once (such as arms and legs). It is defined as a way to calculate the rotations of a chain of bones so that the end of the chain reaches a point (in this case a type 2 IK bone). It starts out with a byte[2] (unsigned short in C++) describing the number of IK chains in the model. Then, for each IK chain in the model:

Data type	**Description**
byte[2] (short)	The index of the IK bone that the chain is trying to reach.
byte[2] (short)	The index of the bone at the end of the chain (or "end effector"). This is the bone that we are trying to make reach the destination point.
byte[1] (unsigned char)	Number of links (bones) that are in this chain. This is followed by the links list
byte[2] (short)	Max number of iterations though the chain before determining that the "end effector" cannot reach the destination point. This is used in the CCD (cyclic coordinate descent) IK algorithm.
byte[4] (float)	Value that represents the maximum angle between any bone in the chain and the destination point in the positive or negative direction
byte[2] (short)	Links List. This is a list of numbers that point to indices in the bone list of the bones that are in this chain. The list starts with the bone closest to the "end effector" and moves up the chain. The length of this section is equal to the number of links.

Face Morph information Edit

This section stores information about the face morphs in the model. These are used to "morph" parts of the model (mouth, eyelids, etc). These are NOT the same as bones! Note that this section starts with a "base" face that contains ALL the vertices that will be used IN TOTAL for ALL the rest of the morphs in the model.

This section starts out with a byte[2] (unsigned short in C++) describing the number of face morphs in the model. Then, for each face morph in the model:

Data type	**Description**
char[20]	The name of the face morph
byte[4] (unsigned long)	The number of vertices that are effected by the face morph
byte[1] (unsigned char)	Face Type. There are 4 types: Eyebrow(1), Eye(2), Lip(3), or Other(0) (look in PMD Editor)
VERTEX LIST	**SEE BELOW**

Face Vertex List Edit

This section comes right after the "Face Type" entry for each face morph data block. It contains the list of vertices that are effected by the face. The position entries represent the maximum possible coordinate that the current face morph can move the vertex to. The coordinates in between are calculated with linear interpolation using the weight value (0.0-1.0, obtained from the VMD motion data).

Data type	**Description**
byte[4] (unsigned long)	Index in the BASE FACE vertex array: the FIRST Morph called "base"
byte[4] (float)	X-coodinate
byte[4] (float)	Y-coordinate
byte[4] (float)	Z-coordinate

Display Name Information Edit

This section describes the display names and groups for the model. This section is only used in MMD to group the bones and face morphs in the dopesheet on the left.

Faces Edit

Indices for the face morphs which should be displayed in the "Facial" section in MMD. This section starts with a byte[1] (unsigned char in C++) representing the number of face morphs to display in the "Facial" section of MMD's dopesheet. Then, for each face morph to display:

Data type	Description
byte[2] (unsigned short)	Index in the face morph list of the face morph to display

Bone Group Names Edit

Names for the bone groups to display in MMD's dopesheet. This section starts with a byte[1] (unsigned char in C++) representing the number of bone groups to create in MMD's dopesheet. Then, for each group:

Data type	Description
char[50]	Name of the group in Japanese

Displayed Bones Edit

Indices in the list of bones for which bones should be placed within which groups in MMD's dopesheet. This section starts with a byte[4] (unsigned long in C++) representing the total number of bones which should be displayed in all groups. Then, for each displayed bone:

Data type	Description
byte[2] (short)	Index in the bone list representing which bone to display
byte[1] (unsigned char)	Index in the list of bone groups representing which group to place this bone in

This is the end of the base model format. After this is the physics information, english model information, and replacement toon texture information. This is the end of the file for models meant for older versions of MMD, but some models do not use this extended format. Some models may include any number of these 3 sections, but not include the others. Do not assume that the model will contain all the following sections if it contains one of them.

English Model Information Edit

This section describes the english information for the model. It contains the model information in english, face and bone names in english, and bone group names in english. The section starts with a byte[1] (unsigned char in C++) representing whether the model has english information in it. It then follows this setup:

English Model Information Edit

The model information in english

Data type	Description
char[20]	Model Name in english
char[256]	Model Comment in english

English Bone Names Edit

The bone names in english. For each bone in the model:

Data type	Description
char[20]	Bone Name in english

English Face Names Edit

The face morph names in english. For each face in the model NOT INCLUDING THE BASE FACE:

Data type	Description
char[20]	Face morph name in english

English Bone Group Names Edit

The bone group names in english. For each bone group:

Data type	Description
char[50]	Bone group name in english

Toon Texture Information Edit

This section describes which toon texture should be changed from the default, and what they should be changed to. By default, the 10 toon textures MMD uses are toon01.bmp, toon02.bmp, toon03.bmp...toon10.bmp. However, models can replace these with their own toon textures. For each of the 10 toons:

Data type	Description
char[100]	File name of the new toon texture (if there is one)

Rigid Body Information Edit

This section describes the rigid bodies in the model, which are used for physics calculation. MMD uses the Bullet Physics Engine, so the PMD models are formatted so that the information here can easily be plugged into Bullet. As such, everything here will be described in terms of the Bullet API. This section starts out with a byte[4] (unsigned long in C++) representing the number of rigid bodies in the model. Then, for each rigid body:

Data type	Description
char[20]	Name of the rigid body
byte[2] (unsigned short)	ID of the bone that is effected by or effects this rigid body
byte[1] (unsigned char)	Collision group (Bullet Physics specific)
byte[2] (unsigned short)	Collision group mask (Used to narrow down the collision group even more. Bullet Physics Specific)
byte[1] (unsigned char)	Shape of the rigid body. It can be sphere, box, or oval (capsule). Once again, this is specific to Bullet Physics
byte[4] (float)	Width of the shape (valid for all shapes)
byte[4] (float)	Height/Radius of the shape (valid only for sphere or capsule shapes)
byte[4] (float)	Depth of the shape (valid only for box shape)
byte[4] (float)	X-coordinate of the shape/body (relative to the bone it is attached to)
byte[4] (float)	Y-coordinate of the shape/body (relative to the bone it is attached to)
byte[4] (float)	Z-coordinate of the shape/body (relative to the bone it is attached to)
byte[4] (float)	X-rotation of the shape/body
byte[4] (float)	Y-rotation of the shape-body
byte[4] (float)	Z-rotation of the shape/body
byte[4] (float)	Mass of the body
byte[4] (float)	Linear Dampening Coefficient (Bullet Physics Specific)
byte[4] (float)	Angular Dampening Coefficient (Bullet Physics Specific)
byte[4] (float)	Restitution (recoil) Coefficient (Bullet Physics Specific)
byte[4] (float)	Friction Coefficient
byte[1] (unsigned char)	Body type (See below)

Rigid body types:

- Kinematic: Body moves with the bone only and is not effected by the simulated physics
- Simulated: Body effects the bone, not the other way around. Body is calculated using the physics engine
- Aligned: Body is effected by the bone and physics engine

Physics Constraint Information Edit

This section describes the constraints in the model. These are, once again, specific to the Bullet Physics engine. MMD calculates it's physics dynamically when the model is using by using constraints. Certain rigid bodies (Kinematic type rigid bodies) move only with bones, and are not effected by the physics engine in terms of movement. Bullet then uses these constraints between bodies to simulate the physics of other bodies. For an example, think of Miku's hair. Her head is attached to a Kinematic rigid body so that it can move with the motion freely (last time I knew, your head doesn't go flying around when you walk forward). Her hair is attached to multiple simulated rigid bodies, which *are* effected by the physics engine. The constraints are put in place to make sure that when the rigid body on her head moves (say, she is walking forward), her hair stays attached to her body. That way, her hair is able to freely move, but stay attached to the rest of her body. A constraint exists between 2 rigid bodies only. These can also be called Joints (found in the last tab of PMD Editor, the direct translation of the menu is "Joint"), as they are placed in a position (usually the point where the 2 bones or rigid bodies touch) and have rotation and position limits.

This section starts out with a byte[4] (unsigned long in C++) representing the number of constraints in the model. Then, for each constraint:

Data type	Description
char[20]	Constraint Name
byte[4] (unsigned long)	Index in the list of rigid bodies of the first body effected by this constraint
byte[4] (unsigned long)	Index in the list of rigid bodies of the second body effected by this constraint
byte[4] (float)	X-coordinate of this constraint (relative to first rigid body)
byte[4] (float)	Y-coordinate of this constraint (relative to first rigid body)

byte[4] (float)	Z-coordinate of this constraint (relative to first rigid body)
byte[4] (float)	X-rotation
byte[4] (float)	Y-rotation
byte[4] (float)	Z-rotation
byte[4] (float)	X-position lower limit
byte[4] (float)	Y-position lower limit
byte[4] (float)	Z-position lower limit
byte[4] (float)	X-position upper limit
byte[4] (float)	Y-position upper limit
byte[4] (float)	Z-position upper limit
byte[4] (float)	X-rotation lower limit
byte[4] (float)	Y-rotation lower limit
byte[4] (float)	Z-rotation lower limit
byte[4] (float)	X-rotation upper limit
byte[4] (float)	Y-rotation upper limit
byte[4] (float)	Z-rotation upper limit
byte[4] (float)	X stiffness (for spring constraint)
byte[4] (float)	Y stiffnedd (for spring constraint)
byte[4] (float)	Z stiffness (for spring constraint)
byte[4] (float)	X-rotation stiffness (for spring constraint)

- **END OF THE FILE****

Category: Articles /About MikuMikuDance