PMVIS

Partitioned Mesh Visualizer

A program to visualize partitioned meshes

Version 1.09

Bilgehan Uygar Öztekin, George Karypis, Vipin Kumar

University of Minnesota, Department of Computer Science /
Army HPC Research Center
Minneapolis, MN 55455

{oztekin, karypis, kumar}@cs.umn.edu

CONTENTS

1 Introduction

Partitioned meshes are commonly used in high performance computing to distribute the computation into a number of processors. PMVIS is developed to view these meshes, allowing users to examine partitions by means of zooming, rotations, translations, scaling, and applying a clipping cube. Variable transparency, hiding a subset of partitions, changing explosion factors, centering on different partitions are some of the capabilities of the program.

2 What is PMVIS

PMVIS is a program for viewing partitioned unstructured meshes consisting of triangular, quadrilateral, tetrahedral, or hexahedral elements. It has been optimized for time critical operations in the preprocessing and displaying stages.

PMVIS is written using OpenGL and Glut hence it should run on any machine that has proper OpenGL and Glut support. However its performance highly depends on the graphics subsystem and the amount of memory dedicated to it, and for some systems it may not function at all for a specific dataset if the video memory size is not sufficient to hold it.

3 Invoking PMVIS

PMVIS consists of a single executable file and an optional file, partdmesh included in the Metis distribution. A typical dataset for PMVIS will have three files: a node file, a connectivity file and a partition file. If they are called data.xyz data.con and data.part respectively, and if the data consists of tetrahedral elements with C/C++ indices used PMVIS can be run using the following command line:

```
pmvis -n data.xyz -c data.con -p data.part -g tet -o 0
```

PMVIS parses the command line parameters and tries to initialize its data structures accordingly. Before initializing any graphics windows and any user-interface support, it reads the input files, allocates required memory and processes the data. It tries to catch and report any possible problems during the initialization process using exceptions. It will also give a summary of the data read (number of nodes, elements, polygons, and partitions). It is always a good practice to check this summary to see if the data has been properly read.

If any error occurs during the initialization stage, an error string will be written to standard output and the program will terminate. If no parameter is given or if an invalid parameter is passed, a summary of possible parameters are written before the program terminates.

Possible command-line parameters that can be used will be explained in detail in the next section.

3.1 Parameters

PMVIS has a number of parameters having both short and long notations, more precisely one can use any string starting with the short version included in the first n characters of the long version. For example if a parameter's short version is quad and the long version is quadrilateral, any string s such that s >= "quad" and s <= "quadrilateral" is accepted as quad. Thus in most of the cases writing a few of the starting characters of a parameter will suffice to identify that parameter.

Here is the help screen given if no parameters are passed:

```
Usage: pmvis parameters
Parameters are:
  -n[odes]         node_file           -> specifies node file           (required
  -c[onnectivity]  connectivity_file   -> specifies connectivity file   (required
  -p[artition]     partition_file      -> partition file                (optional)
  -m[etis]  number_of_partitions  -> use metis to build the partitions (optional)
  -k[eep]      -> force to keep intermediate elements (default: only boundaries)
  -o[ffset]  n -> n=0:use C/C++ style, n=1:use Fortran sytle indices (default:0)
  -g[eometry] tri[angle] quad[rilateral] tet[rahedral] hex[ahedral]  (def:tets)
```

Characters between brackets denote the redundant ones and can be omitted. For example for the case of –geometry, which is denoted by "-g[eometry]" the short version becomes "-g", the long version becomes "'-geometry", and as an example "-geom" will also be recognized as the geometry parameter.

Here is the list of parameters and their explanations:

-nodes node_file:

Specifies the coordinates file. (Explained in section 5.2)

-connectivity connectivity_file:

Specifies the connectivity file. (Explained in section 5.2)

-partition partition_file:

Specifies the file containing the partition vector. (Explained in section 5.2)

-keep:

This option overrides the default behavior of removing intermediate elements. If it is specified, all original elements will be kept. If it is not given, only the boundary elements will be kept and displayed. We recommend using this parameter only if intermediate elements are important in visualization, since by removing non-boundary elements, number of polygons that needs to be stored and drawn is reduced considerably.

-offset n:

Specifies the offset value used while processing the connectivity file. n can be either 0 or 1. This parameter is added to allow both C/C++ and Fortran array convention. If offset is 0, C/C++ convention is assumed (first node has index of 0), if it is 1, Fortran style indices are assumed (first node has index of 1). Default behavior is to use offset of zero, C/C++ convention.

-geometry geometry_type:

Specifies the geometry type, which can be either one of these: triangle, quadrilateral, tetrahedral, and hexahedral. If it is omitted, tetrahedral geometry is assumed.

-metis number_of_partitions:

This option can be used to call Metis to obtain the partitions. PMVIS assumes that an executable “partdmesh” is in the current directory and tries to call it to obtain the partition vector with specified number of partitions. Note that partdmesh is not part of PMVIS, it is part of Metis, and must be compiled separately. Win32

distribution will include partdmesh as an executable but for other platforms Metis must be compiled to obtain it.

4 User Interface

PMVIS uses both mouse and keyboard as inputs. Right mouse button can be used to start the menu in which one can display a summary of commands anytime by selecting help. Here is the help page printed to console when help command is selected:

```
lb, rb and mb denote left, right and middle mouse buttons respectively.
lb            -> rotation
lb and rb     -> y dir: zoom in/out, x dir: adjust clipping cube size
mb            -> translation
mb and rb     -> y dir: move clipping cube in/out, x dir: clipping cube size
ctrl+lb       -> same as mb but translation state is locked until lb release
shift+lb      -> same as lb and rb but state will be locked until lb release
ctrl+shift+lb -> same as mb+rb but state will be locked until lb is released
arrow keys    -> rotation,                pageup/pagedown    -> zoom in/out
Ctrl and shift combinations are useful if the mouse does not have 3 buttons

[n]          [enter*] -> center on partition n (use 0 for the whole object)
's'[n1]      [enter*] -> select/desellect partition n1 ex:"s3" sel/desel 3rd
's'[n1];[n2] [enter*] -> select partitions in range n1 to n2
ctrl+[function key]   -> store current selection to function key (F1 to F8)
[function key]        -> recall previously stored partition selection
(*)Program will automatically feed enter after a timeout if not pressed

't' -> apply current transparency value to selection
'o' -> make current selection opaque
'h' -> hide/unhide current selection
'r' -> randomly shuffle colors
'c' -> clipping mode on/off
'+' -> select all,  '-' -> select none,  '*' -> invert selection
```

4.1 Using the mouse

The default function of the right mouse button is popping up the menu. Left button is used to switch to rotation mode, in which the right button becomes overridden and is now

used to switch to “zoom” / ”adjust clipping cube size” mode as long as the left button is pressed. If right button is pressed while left button is down, y direction is used for zooming in and out, and the x direction is used to adjust the clipping cube size.

Middle button is used for translation, if pressed, x direction moves the object in the left-right direction, and the y direction will move it in the up-down direction relative to the current position and direction of the camera and its target. Note that in all cases the clipping cube will be where the camera is aiming. If middle button is pressed, right button’s function changes similar to the left button case. In short x direction of the mouse will behave exactly the same as previous case i.e. adjusting the clipping cube size, and the y direction will look like it is zooming but unlike zooming option, the distance between the camera and the clipping cube is fixed. This feature is used to translate the clipping cube in and out of the screen relative to the camera. It is possible to move the clipping cube anywhere in the mesh by using middle button and middle and right button combinations.

Clipping cube is used to draw only parts of the mesh that is inside the cube. This feature allows users to concentrate on only a portion of the mesh. Moreover it doubles as a reference point about which rotations occur. Camera will always center on the clipping cube during all operations. By pressing ‘c’ or selecting clipping mode from the menu, user can turn clipping feature on and off.

Note that for compatibility with two button mice we also allow using shift and control keys in conjunction with the left button. Ctrl + left button will behave as if middle button is pressed and shift + left button is used to simulate right click. In short left button will behave as it was before, shift + left button will behave as if right button was pressed while left button is down, ctrl + left button will behave as if middle button was pressed while left button is down, and ctrl + shift + left button will behave as if middle button and right button are pressed.

Note that for the case of zooming or translating in out, and “adjusting clipping cube size”, there is a threshold in y movement vs. x movement of the mouse. If for example the user is using the mouse to zoom in and out, the mouse should mostly go in forward/backward direction (y direction) but obviously small variations in x direction may occur. PMVIS compares movement in y direction and x direction, if change in y direction is larger than a constant factor of the change in x direction the latter is ignored. Which basically means that during zooming in/out and translating in/out, clipping cube size will not be modified by accidental variations in x direction.

Cursor keys can also be used to rotate the object, and page up and page down keys can be used to zoom in and out. But there is no equivalent of translation using the keyboard in this version.

4.2 Using the keyboard

PMVIS is able to select deselect partitions, form partition groups, center on a given partition, apply transparency value to selected partitions and hide selected partitions.

In order to move the center (thus the clipping cube) to a partition one may just write the partition number. Note that there is a timeout mechanism after each key press, which automatically feeds enter after about 500 ms. For example, to center on the 15th partition, one can press “1” and “5” consecutively, after that, one could either press the enter key or just wait 500ms to center on the 15th partition. If the user is too slow to press “5” after pressing “1”, timeout will occur and the 1st partition will be selected instead of 15th, in fact, first, the 1st partition will be selected, then the 5th one. This is a design choice and we have chosen to go this way in order not to be forced to press enter each time and use leading zeros for partitions with small numbers. If 500ms is too small, value of the variable “timeout” can be set to the value desired in seconds by changing the code and recompiling. Setting this value very high will effectively disable auto-feeding feature.

The “s” key is used to select or deselect a single partition. Pressing the “s” key and then writing the number of the partition will toggle selection status of that partition. Note that this works exactly the same way as centering on a partition i.e. auto-feeding enter is in effect. For example pressing “s”+”1”+”5” will select the 15th partition if it was not selected, and deselect it if it was already selected. Note that selected partitions are drawn with a bright white color.

To select a range of partitions one can use the “s” key in conjunction with a range. Ranges are indicated by using a “;” or “:” between two numbers. For example “8:13” will select all the partitions from 8 to 13 inclusive. If the first number is omitted, selection will start from the first partition, and if the second number is omitted selection will go until and including the last partition, thus for example, “s:” will select all of the partitions, “s:10” select the first 10 partitions, and “s5:” will select all of the partitions with numbers greater than 4.

In addition to “s” selection, “+” can be used to select all of the partitions, “-“ to deselect all, and ‘*” can be used to take the complement of the current selection.

Function keys F1 to F8 can be used to store current selection and retrieve it back later. Ctrl + Function key will store current selection to the selected function key, and pressing the function key will retrieve the selection previously stored.

There are a number of operations that can be done on a partition selection:

- Transparency/opacity: Pressing the “t” key while a selection is in effect, will change the transparency value of the partitions selected using current transparency value (see transparency menu, to learn how to change the default value). Pressing the “o” key will make current selection opaque, supplying a transparency value of 1.0.
- Hide/unhide: Pressing the “h” key will toggle hide / unhide selected partitions. For example, if the user is only interested in 4th and 7th partitions. He/she can

select them by entering “s4”, “s7”, press the “*” key to take the complement, and press the “h” key to hide all but 4^{th} and 7^{th} partitions.

- Pressing the “r” key will randomly recolor the partitions. This feature is useful in changing colors especially if only a subset of the partitions is shown. PMVIS automatically tries to get equidistant colors by using HSV color scheme but if the number of partitions is large, obviously some of the colors will be very close. Randomly reassigning colors may result in a selection of distant colors for neighboring partitions of interest.

4.3 Using the menu

The menu is accessed by right clicking on the PMVIS window.
Here is the list of menu options with explanations:

- Model: Options are: Wireframe, black polygons, polygons only and outlined polygons. The default one is outlined polygons, in which each polygon is outlined with a slightly darker color of its color. Wireframe will only draw the contours of the polygons, and polygons only will draw polygon surfaces but not the contours.
- Lines: Three options: Antialiasing off, low quality, best quality, and “don’t care”. Antialiasing is a method of rendering lines in order to minimize aliasing effects (patterns that occur due to limitations in sampling). OpenGL supports low quality, best quality and “don’t care” quality for antialiasing. Low quality should be faster, best quality will be better but slower, and “don’t care” will leave the decision to OpenGL. Note that the behavior of those selections will depend on the specific OpenGL hardware and drivers used. For some systems, only one method of antialiasing is implemented, thus one would only see the difference between antialiasing off and one of the selected methods.
- Labels: Possible options are: Labels off, preview mode only, and all modes. By default, labels are written only in preview mode i.e. during translation, rotation etc. User can change this behavior using this submenu.

- Preview: Preview mode is used to speed up rotations, translations etc. Moreover it is a nice method of seeing partitions, partition labels, and the clipping cube. We highly recommend usage of the preview mode. But if mesh size is relatively small, and if OpenGL hardware is fast enough, decent frame rates can be obtained even without the preview mode.
- Culling: Culling is a method of eliminating polygons without even checking the depth buffer. It is using normals of the polygons and the location of the camera to determine if a given polygon should be drawn. For convex and closed shapes, culling can be used to increase rendering speed considerably. Front faces and back faces are used to select the faces that should be removed. If the mesh has a fixed convention about how the normal vectors are specified, either front face or back face culling method should work. Moreover if for example front face culling is the one that gives expected results for a given mesh, then the opposite, i.e. back face culling, will have the effect of opening the objects, and seeing the interior.
- Clipping: User can turn on and off the clipping cube feature (menu equivalent of the “c” key). If clipping is on, only the portions of the mesh inside the clipping cube will be rendered.
- Explosion: Explosion is a method of separating partitions from each other. PMVIS achieves explosion by shrinking each partition towards the center of its bounding box. This method is quite fast and can be implemented by using translation and scaling in OpenGL system, which is in general supported by hardware. Explosion factor determines how much scaling will be done. If 1.0 is selected, no explosion will occur. If for example, 0.3 is selected, each partition will be scaled by a factor of 0.3. A specific explosion value can be given using console input feature in the menu.
- Transparency: PMVIS supports two kinds of transparency methods. The first one is “one minus alpha” and the second one is “one” which corresponds directly to OpenGL transparency functions. The first one depends on the order of which the primitives are drawn and produces better results, but the method requires sorting all primitives from back to front and drawing in this order. This operation is too costly and requires changing the display lists. As a suboptimal solution to this

problem, PMVIS keeps a display list for each partition, sorts each partition on the fly according to the center of their bounding box from back to front, and renders them in this order. This would give decent results for most cases but it is not the best solution in terms of quality. The second function does not depend on ordering but does not exactly correspond to intuitive transparency notion. If the inside elements are removed and if the partitions are well separated, the first method should give decent results. Transparency can be enabled and disabled from this menu, and the default transparency value that will be used by the “t” key until a further change can be set. Note that while changing current transparency value, if a subset of the partitions is selected, their transparency values will be automatically updated to the new value. If this behavior is not desired, one must deselect all partitions before changing current transparency value. Note that custom transparency values can be given using console input option of the menu.

- Colormap: Color map can be set to its default value or can be randomly reselected. The latter is equivalent of pressing the “r” key.
- Console input: This is used to input custom values for some options, which are: Center position (position of the clipping cube’s center), camera position, clipping cube size, transparency factor and explosion factor. If any one of them is selected, PMVIS will require input from the console. Note that if a selection is in effect and if transparency factor is updated, the new value will also be applied to current selection just like changing the transparency value.
- Help/About: Help prints a one page summary of keys and mouse bindings to the console. About, prints version number etc.
- Quit: This menu choice as well as the “q” and “x” keys can be used to close the program.

5 File Formats and Conventions

5.1 Node numbering and element types

PMVIS supports four element types: Triangles, quadrilaterals, tetrahedrons, and hexahedrons. PMVIS only works with triangles and quadrilaterals as primitive elements. For triangles and quadrilaterals, elements and the primitives passed to the OpenGL system are the same but in the other two cases we have to convert each element into a number of primitives. Each tetrahedron is converted to four triangles, and each hexahedron (brick) is converted to six quadrilaterals. Note that the ordering of vertices are done while preserving the counter-clockwise rule for the normal vectors in order to obtain a consistent back face and front face culling. There is only one way of ordering a tetrahedron but there are different ways to do it for a hexahedron. The following figures show the node numbering convention for each type.

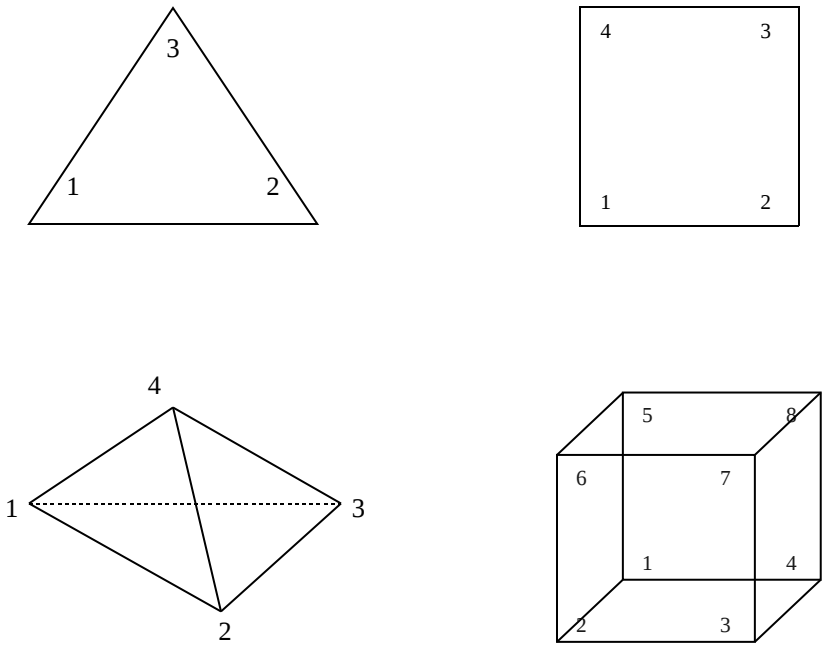

Figure 1: Node numbering convention for triangles, quadrilaterals, tetrahedrons, and hexahedrons

5.2 File formats

PMVIS accepts ASCII files (text files). Connectivity file and partition vector file are integer files and the node file (or coordinate file) is a file of floats. Basic data types used and assumed in PMVIS are integers and floats. If this choice is not convenient for any application, source code can be modified for the appropriate types easily by just changing two type definitions near the beginning of the source code:

```
#define real float;
#define index int;
```

PMVIS requires the data in a specific format. All meshes will require at least two files:

1) Coordinates file, which will have the x, y, and z coordinates for each node.
2) Connectivity file, which tells which nodes belong to a given element.

For example let us suppose we have the following files:

Coordinates file (node file)

```
0.5 0.5 0.1
1.2 3.4 5.6
3.5 –0.1 0.2
7.1 0 –2.2
```

Let us further suppose that we have two triangles in the mesh, so we will have two entries in the connectivity file for example if we have (assuming Fortran numbering):

```
1 3 2
2 4 1
```

The first triangle will have the 1st 3rd and the 2nd nodes in the node file, and the second triangle will have the 2nd 4th and the 1st nodes. The vertices of the first one will be (0.5, 0.5, 0.1), (3.5, –0.1,0.2), and (1.2, 3.4, 5.6) and so on.

Coordinates file will contain three times the number of elements that are all floats. It's contents will be like $x_1\ y_1\ z_1\ x_2\ y_2\ z_2\ \ldots\ x_n\ y_n\ z_n$ for all n elements.

Contents of the connectivity file are integers; more precisely they are indices in the coordinate file. Connectivity file must contain number of elements times number of nodes per element, many integers. Number of nodes per element is 3 for triangles, 4 for quadrilaterals, 4 for tetrahedrons, and 8 for hexahedrons. For example if we have n quadrilateral elements, contents of the connectivity file would be:

$i_{11}\ i_{12}\ i_{13}\ i_{14}\ i_{21}\ i_{22}\ i_{23}\ i_{24}\ \ldots\ i_{n1}\ i_{n2}\ i_{n3}\ i_{n4}$

Note that, white space characters such as tabs, line feeds, spaces etc. can be placed anywhere between the entries thus, a good practice is to use a line per node in the coordinate file and a line per element in the connectivity file.

Node files and connectivity files are similar for the case of other element types. If for example we have a tetrahedral mesh of two elements. We can have the following as connectivity file:

2 3 4 1
1 2 4 5

This states that the first tetrahedron has as vertices the 2nd, 3rd, 4th, and the 1st entries in the node file and the second one has 1st, 2nd, 4th, and 5th nodes. (numbering of nodes are as given in figure 1 for each element type).

Note that not all of the node file's contents need to be used in the connectivity file. This allows a subset of the mesh to be examined by forming a new connectivity file (a subset of the original file) and using the same node file.

Optionally a partition file containing the element vise partition vector can be given as parameter using the "-partition" switch. The file should contain number of elements integer entries. Each entry specifies which partition a given element belongs to. Note that the numbers can be arbitrary, they don't need to start at a given number and they don't need to be consecutive. PMVIS will automatically sort them and reassign numbers starting from 1. As an example if we have 4 different partitions the following partition vector: 4 2 1 1 2 3 will assign first element to 4^{th} partition, second to 2^{nd} one, third to 1^{st} partition and so on. If no partition file is given, it is assumed that all of the elements belong to the first partition.

6 Hardware Requirements

PMVIS is written in C++ using OpenGL and Glut libraries. It is quite portable and should work in many platforms such as various flavors of Unix, Windows, OS/2, and MacOS. It is tested for SGI Iris 5.3, Windows 98/NT, Linux using Mesa library and it should work on any platform with a fairly up to date C++ compiler, OpenGL and Glut. Although Glut does not allow advanced user interface features, it is highly portable, making OpenGL and Glut combination one of the good choices for cross-platform program development.

6.1 Performance

In order to understand PMVIS's performance and limitations one must know how PMVIS works. During the initialization stage, each element is first converted to either triangles or quadrilaterals and fed to the graphics subsystem via display lists. Graphics memory that the program requires is directly proportional to the size of the display lists. For example if we have 100 tetrahedral elements, number of vertices that will be used is about 100*4*3 (Three vertices for each triangle). They are stored as GLfloats and memory required should be about a constant times the number of floats required to store the mesh plus a small constant. Current version of the program does not compress the elements into straps or fans, which may decrease the memory requirement roughly by two, depending on the nature of the mesh, but this would require additional preprocessing and we are not aware of a cheap and efficient algorithm yielding an optimal solution. We are however removing in an efficient manner the entire set of interior elements (non-boundary elements) and work only on the boundaries if the "-keep" switch (see parameters) is not present. For example, in our tests a mesh consisting of 30000 tetrahedral elements was simplified down to about 5000 triangles as opposed to 120000 if interior elements are removed. Note that this feature is quite an improvement in terms of speed and memory but it will only show elements on the boundaries, fortunately this is what people is usually concerned with for most practical purposes.

We tested PMVIS with datasets of various sizes with different number of partitions. Number of partitions does not seem to be a bottleneck in performance up to a reasonable number of partitions; we obtained acceptable frame rates (a few frames every second) on a Pentium III 600 with a 3DFX Voodoo3 graphics card using a mesh of 30000 tetrahedral elements and 1000 partitions (about 39000 triangles in total after removing inside elements). With the same setup, but using 100 or less number of partitions, we obtained real-time frame rates. Complexity of the updates after the initialization step is practically O(n * p) if no transparency is used, and it is roughly O(n * p * log p) if transparency with one minus alpha function is used, where n is the number of polygons and p is the number of partitions. Note that if the dataset used is very large (millions of elements) preprocessing time may take up to a few minutes but this is unavoidable to obtain decent interactivity afterwards.

6.2 Compiling

PMVIS requires OpenGL and Glut libraries. For information about availability for a specific platform and specific graphics subsystem one may check http://www.opengl.org. For Win32 platform a precompiled executable is supplied. For other platforms, PMVIS must be compiled. For example for SGI Iris, command line used for compilation is:

```
CC pmvis.cpp -exceptions -lglut -lGLU -lGL -lXmu -lXext -lX11 -lm -o pmvis
```

For our Linux test system using Mesa (http://www.mesa3d.org, a free OpenGL work-alike library), command line used to compile is:

```
g++ pmvis.cpp -exceptions -L/usr/X11R6/lib -lglut -lMesaGL -lMesaGLU -lX11 -lm -lXext -lXmu
-lXi -o pmvis
```

PMVIS is written in C++ and can use both old headers and new headers. It uses exceptions to catch file input output errors, memory allocation errors and similar run-time errors, but if a specific platform does not have a compiler supporting exceptions or the new header files, it is still possible to compile the code with limited error checking.

Three macros are used to change the behavior of the compiler:

#define USE_NEW_HEADERS
#define USE_EXCEPTIONS
#define USE_NAMESPACE

If USE_NEW_HEADERS is defined, PMVIS will use new C++ headers, If USE_EXCEPTIONS is defined, try and catch statements will be included in the code, and if USE_NAMESPACE is defined, (goes together with new headers) “std” namespace will be used throughout the code. To disable any of them, one can remove the associated macro definition near the beginning of the code.

6.3 Contact Information

PMVIS has no know bugs but this does not mean that we have found all of its bugs and fixed them. If you encounter any bugs, please feel free to contact us via email (Bilgehan Uygar Oztekin, oztekin@cs.umn.edu), giving a brief description of the problem. Note that there are some features of PMVIS that are not documented on purpose such as binary file support, which has portability and compatibility issues. Users may use this feature at their own risk. They are not tested and may not exist in possible future versions.

Most up to date information about PMVIS can be found in PMVIS’s home page, which is located at http://www-users.cs.umn.edu/~oztekin/pmvis/

7 Copyright