

SGE/Mesa Reference Guide

Version 1.0

Kenneth Perrine, Pacific Northwest National Laboratory

June 20, 2001

About SGE/Mesa

SGE/Mesa is a back-end driver for the Mesa OpenGL implementation to allow parallel applications to render OpenGL graphics and output them to the IBM Scaleable Graphics Engine (SGE). Multiple nodes share a region of the output which is currently automatically assigned. SGE/Mesa also supports the use of multiple threads to render to multiple frames simultaneously.

SGE/Mesa has been tested with Mesa version 3.4.2, and it is intended that it will compile with future versions with little modification.

SGE/Mesa can take advantage of multiple nodes and/or threads by splitting up color buffers among nodes and threads. Multiple frames of an animation sequence can simultaneously be rendered. In addition, parallel performance during the tunneling stage (when data is transferred to the SGE) can be achieved if a color buffer is distributed across more than one node.

GLUT, a window manager library for simplifying the use of OpenGL implementations is modified to automatically initialize SGE tunneling support for windows and to also properly handle events for the SGE. Events generated from an SGE window are distributed to all nodes in the group used to initialize GLUT. The GLUT pop-up menus are supported in the parallel environment and their events are distributed to all nodes. This modified GLUT library is called SGE/GLUT.

Since all required SGE functionality and parallel event handling is managed in the libraries, many codes which utilize OpenGL and GLUT libraries will compile with the SGE/Mesa and SGE/GLUT libraries and utilize the SGE in the parallel environment with no code modifications. Running code which is designed for a single workstation in parallel causes the SGE/Mesa library to automatically split the display into regions (with each region being maintained by a node) and causes the renderer to render only the geometry which falls within the region that corresponds with the node. Rendering regions and associated clipping planes on each node are automatically set in the `glViewport`, `glFrustum`, and `glOrtho` Mesa function calls.

The SGE/Mesa library also has a function call which creates rendering threads. This allows multiple processors on each node to simultaneously work on rendering tasks. Each thread can render a single frame in a series or render a region of one or more frames.

It is planned that a future generation of SGE/Mesa will incorporate compositing functionality which will allow multiple threads working on one frame to each be responsible for only a portion of the geometry transformation and rendering. Separate images would then be composited together (by keying on the background color or utilizing the depth buffer) before being tunneled or modified.

SGE/Mesa Programming Considerations

The SGE/Mesa driver and SGE/GLUT have been made to provide minimal impact to existing code which utilizes GLUT and OpenGL calls. Many of the example programs which come with Mesa 3.1 do not require modification to compile and run in the parallel environment with SGE/Mesa. There are, however, a couple of requirements and considerations:

The `glutInit` function must be called before any other GLUT function calls are made, except for the SGE/GLUT `glutSGESetMPIGroup` function. The `glutInit` function sets up variables specific to SGE/Mesa and also initializes the MPI library. If the `glutInit` function is not called, most likely an MPI error results saying that MPI had not been initialized. Note that it is possible to initialize MPI in the source code and then call `glutSGESetMPIGroup` before calling `glutInit`. This keeps `glutInit` from attempting to initialize MPI a second time.

`glutSwapBuffers` or `SGEMesaSwapBuffer` must be called in order to transfer graphics data to the SGE. Without any call to a swapping function, no data transfer takes place. The `glFlush` function does not cause a buffer swap, and requests for direct rendering are ignored.

Other unresolved issues exist as identified in the “Bugs/Improvements” section.

A debug mode is available for allowing the borders of regions to be seen. (This is done by reducing the size of regions tunneled to the SGE by one pixel). To enable this, set the “`SGEMESA_R`” environment variable. (The value of the variable may be anything).

SGE/Mesa Multithreaded Considerations

All files in the “demos” directory can run in parallel but they don’t support multiple rendering threads. In configurations where one process is run on each node and each node contains more than one processor, the remaining processors are not utilized unless threads are created in the application. Additional rendering threads may be used to render to multiple frames or to render to separate regions of common frames. A separate rendering thread may also be used to allow rendering to take place while a tunneling operation is being performed. The `SGEMesaCreateThread` function creates a separate rendering thread which creates its own Mesa rendering context and allows rendering to take place to a separate buffer (for multi-frame rendering) or to a region of a shared buffer.

The use of multiple threads requires a synchronization mechanism to be in place to prevent variables utilized during rendering from being unexpectedly changed by event handler functions. A queue mechanism is provided by SGE/Mesa to store a void pointer in the main thread and retrieve the pointer in the rendering thread when the rendering thread is ready. An optional destructor may be supplied with the pointer when it is submitted to the queue so that it may be automatically destroyed when it is no longer needed.

See the “agears” and “agloss” demos in the “sgedemos” directory for examples of adapting two of the Mesa demos to run with multiple rendering threads. The “amolref” demo creates a visualization of the molecule specified on the command-line. (Molecules are stored in the “sgedemos/molfiles” directory). Be sure to try right-clicking to get menu options and left-click-dragging to move the molecule around.

To experiment with different configurations of these demonstrations, command-line parameters can control the distribution of regions and frames.

The `-a #` parameter (consisting of dash, a, space, number) sets how many simultaneous frames exist on each node. The number 8 is default; however, the delay between user input and display increases and becomes annoying after 4. Setting this to 1 demonstrates rendering of graphics data while tunneling takes place, which is some improvement over the default approach (as seen by the “demos” applications) where all rendering blocks when tunneling.

The `-b #` parameter sets how many regions are assigned per node per frame. For example, if `-b` is 2, then on each frame, the node is responsible for 2 of the regions. This is 1 by default.

The `-c #` parameter sets how many nodes handle the rendering of each frame through subdivision of regions. The `-c` parameter must be a divisor of the total number of nodes, and is the number of nodes by default. If `-c` is 4, for example, then each frame is rendered across 4 nodes. The number of subdivisions for each frame is this value multiplied by the `-b` parameter.

If you’re running a 4-node job, and you want to have each node render a full frame so that four frames at a time are displayed flipbook-style, you would run with:

```
-a 1 -b 1 -c 1 (for 1 thread per node rendering all of 4 frames)
```

If you’re running a 1-node job and you want to divide the output into 4 regions, then you would run with:

```
-a 1 -b 4 -c 1 (for 4 threads on the node, rendering all of 1 frame)
```

If you’re running a 4-node job and you want to render 8 frames and divide the output into 4 regions per frame, you would run:

```
-a 8 -b 1 -c 4 (for 8 threads per node, each node rendering a quarter of 8 frames)
```

```
-a 4 -b 2 -c 2 (for 8 threads per node, each node rendering a half of 4 frames)
```

```
-a 2 -b 4 -c 1 (for 8 threads per node, each node rendering all of 2 frames)
```

If you’re running a 4-node job and you want to have 4 frames with 4 divisions each:

```
-a 4 -b 1 -c 4 (for 4 threads per node, each node rendering a quarter of 4 frames)
```

An earlier demo application called “agears3d” tests the stereo capabilities of SGE/Mesa, but stereo must be enabled on the SGE in order for this to display both channels.

Additional note: when the SGE GBE (gigabit Ethernet) libraries are used, it is possible to run multiple processes on one node. When the SGE TBS libraries are used for SP switch links, only one SGE process may run on each node because of switch adaptor window limitations. Multiple rendering threads may be created with the SGE/Mesa thread functions, however.

Building SGE/Mesa

Refer to the “SGE/Mesa Installation Notes” for information on building SGE/Mesa and its demo applications. The document also contains troubleshooting items.

Building Applications Which Use SGE/Mesa

When compiling your own applications, you can use the SGE/Mesa Makefile in the “sgedemos” directory for guidance on creating your own makefile. Specifically, the Mesa “GL”, Mesa “GLU”, SGE/Mesa “SGEmesa”, SGE/Mesa “SGEglut”, SGE tunneling libraries, and SGE X11 libraries must be included when creating an application which utilizes SGE/Mesa.

SGE/Mesa Function Reference

The following sections are a function reference for all publicly-available SGE/Mesa function calls. The first section describes functions which are automatically called by the SGE/GLUT and are not needed if SGE/GLUT is used. Note that some functions can only be called by rendering threads as noted whereas other functions can only be called by the main thread.

Low-Level SGE/Mesa Functions

The following SGE/Mesa functions are low-level and are automatically called by GLUT. You can use these functions if you are not using GLUT:

SGEMesaCreateTunnel (Display *display, Window *window, MPI_Comm group, GLboolean cursor_flag)

Parameters:

- display – The current X display structure.
- window – The window to initialize for tunneling.
- group – The MPI group corresponding with all nodes which can perform tunneling operations to the window.
- cursor_flag – GL_TRUE if SGE cursor support will be enabled for the window.

Returns:

- SGEMesaTunnel – The tunnel object which can be used to create SGE/Mesa contexts.

Description:

SGEMesaCreateTunnel submits a window for SGE tunneling. The SGE software creates a tunnel region in the window’s client area which can no longer be written to by X11 calls. The current version of the SGE library does not allow the window to be un-tunneled.

SGEMesaCreateContext (SGEMesaTunnel tunnel, GLboolean stereo_flag, GLboolean alpha_flag, GLboolean depth_flag, SGEMesaContext sharelist)

Parameters:

- tunnel – A window-specific tunnel object returned from SGEMesaCreateTunnel.
- stereo_flag – Set this to GL_TRUE to enable stereo support. Separate buffers are created for left and right channels. Buffer update time doubles when stereo is enabled.
- alpha_flag – Not currently used. In a later version, setting this to GL_TRUE will enable the SGE driver to maintain alpha channel information for use in compositing color buffers before they are sent to the SGE.
- depth_flag – Not currently used. In a later version, setting this to GL_TRUE will enable the SGE driver to maintain depth buffer information for use in compositing color buffers before they are sent to the SGE.

sharelist – Not currently used. In a later version, this will enable information for an existing context to be shared with the new context.

Returns:

SGEMesaContext – The SGE/Mesa context object.

Description:

SGEMesaCreateContext creates a context object for drawing to the window identified by the given tunnel object. Multiple contexts can exist for one window, allowing multiple buffers to be maintained which can be enabled and displayed at any time through the use of SGEMesaMakeCurrent. (The use of multiple contexts on one window has not been tested).

SGEMesaDestroyTunnel (SGEMesaTunnel tunnel)

Parameters:

tunnel – The tunnel object returned from SGEMesaCreateTunnel.

Description:

SGEMesaDestroyTunnel is provided for memory clean-up purposes. At this time, it does not un-tunnel the tunnel object's window. Do not call this if any contexts of this tunnel are in use, otherwise undefined behavior will result.

SGEMesaDestroyContext (SGEMesaContext context)

Parameters:

context – A context object returned from SGEMesaCreateContext.

Description:

SGEMesaDestroyContext destroys a context and frees all associated resources. Currently, this does not automatically stop the rendering threads. Do not call this if any rendering threads utilizing this context are in use, otherwise undefined behavior will result.

SGEMesaMakeCurrent (SGEMesaContext context, GLboolean scanFlag)

Parameters:

context – The SGEMesaContext for the window and graphics buffers to make active.
scanFlag – This forces a window size check to be made so that buffers can automatically be reallocated if the output window size changes.

Returns:

GLint – Returns 0 if there is a failure in memory allocation.

Description:

SGEMesaMakeCurrent activates a context and its rendering threads for use with SGE/Mesa. It also reallocates color buffers if it detects a resize in the window and synchronizes sequence and thread information among all nodes.

If no rendering threads exist (through the SGEMesaCreateThread function), SGEMesaMakeCurrent transparently creates a Mesa context specific to the calling thread (the main thread). This allows the calling thread to then make OpenGL calls. This functionality is provided to allow most OpenGL and GLUT code to run on the SGE and in the parallel environment without coding modifications. This context is automatically

destroyed when `SGEMesaCreateThread` is called, since the main thread cannot be used for rendering and tunneling at the same time. This is a synchronous call.

SGEMesaSwapBuffers (SGEMesaContext context)

Parameters:

context – The `SGEMesaContext` for the window and graphics buffer for swapping.

Description:

`SGEMesaSwapBuffers` transfers graphics from the buffer corresponding with the next sequence to display. In this transfer, the back buffer is swapped to the forward buffer which is then tunneled to the SGE. While tunneling takes place, the rendering threads can write to the new back buffers, unless no threads had been created with `SGEMesaCreateThread`. This function is synchronous and should be called by the main thread. This is automatically called by `glutSwapBuffers`.

Note that this function blocks until all rendering threads corresponding with the sequence number of the frame to display have called `SGEMesaThreadPost`. If no rendering threads had been created with `SGEMesaCreateThread`, then `SGEMesaSwapBuffers` is only in charge of swapping the default rendering context and returns when the tunneling operation is completed.

General SGE/Mesa Functions: Main Thread

The following functions are used in conjunction with SGE/GLUT, and may also be used if SGE/GLUT is not used. These should only be called by the main thread and not the rendering threads if any were created with “`SGEMesaCreateThread`”.

SGEMesaCreateThread (SGEMesaContext context, void (*renderFunction)(void *), void *renderArg, GLint seqNo)

Parameters:

context – The `SGEMesaContext` to which the rendering thread will be associated.

renderFunction – The function which will be called by the new rendering thread. This function is the controller for the rendering thread.

renderArg – A void pointer which is passed to `renderFunction` as an argument.

seqNo – The sequence number for the rendering thread. This can be used to associate this rendering thread with other threads for automatic screen division, or to identify this rendering thread’s output in a series of output frames.

Returns:

GLint – The value of `-1` is returned if a thread is successfully created. If not, `0` is returned.

Description:

`SGEMesaCreateThread` creates rendering threads. During the creation process, a thread-specific OpenGL context is generated and associated with the new thread. The thread can then make OpenGL calls which are independent from calls made by other threads. The new thread is suspended until `SGEMesaMakeCurrent` is called. SGE/GLUT automatically calls this in its event loop. To terminate the rendering thread, let the thread exit `renderFunction`. Call `SGEMesaCreateThread` from the main thread.

SGEMesaQueueStore (SGEMesaContext context, GLint seq, void *element, void (*destructor)(void *))

Parameters:

- context – The SGEMesaContext of interest
- seq – The sequence number belonging to the thread(s) which will retrieve the queue item. The SGEMesaNextQueueSeq function can be used to generate sequence numbers.
- element – A pointer to some data structure which will be retrieved by rendering threads.
- destructor – Pointer to a function which can be called when the element for a sequence number is replaced or expired in the queue. As an example, the free function can be specified if the data pointed to by element was allocated with malloc. Specify NULL to keep a destructor function from being called.

Description:

SGEMesaQueueStore is a tool for passing information from controlling code to rendering threads. Specifically, data intended for threads associated with a particular sequence number can be passed from the main thread to those rendering threads without concurrency or synchronization issues. The main thread uses SGEMesaQueueStore to put the data into the back portion of the queue. The data stays in back portion until SGEMesaSwapBuffers is called and the buffers corresponding with the given sequence number are swapped. The data is moved from the back portion of the queue to the forward portion where it is visible by SGEMesaQueueGet. At this point, the rendering thread(s) for that sequence number begin work on a new frame and SGEMesaQueueGet is called by the rendering thread(s) to retrieve the stored data. The data expires in the queue when the next data item in the back portion of the queue is propagated to the foreground portion. This function does not pass data to other nodes.

SGEMesaNextQueueSeq (SGEMesaContext context)

Parameters:

- context – The SGEMesaContext of interest.

Returns:

- GLint – The sequence number for the next available queue position. The value of -1 is returned if there is no position open in the queue.

Description:

SGEMesaNextQueueSeq returns the sequence number for the next available queue position. Note that sequence numbers corresponding with threads on other nodes are also returned. That means that regardless of how sequences are distributed across nodes, concurrent calls to SGEMesaNextQueueSeq on all nodes return the same value on all nodes. (This is assuming that SGEMesaMakeCurrent had been called to synchronize all nodes with the same rendering information. Note that GLUT automatically calls SGEMesaMakeCurrent).

General SGE/Mesa Functions: Rendering Thread

The following functions are designed to only be called from the rendering thread.

SGEMesaThreadPost (SGEMesaContext context)

Parameters:

- context – The SGEMesaContext of interest.

Description:

Rendering threads call `SGEMesaThreadPost` when rendering for a frame is complete and the frame is ready to be tunneled to the SGE. `SGEMesaThreadPost` blocks until the main thread calls `SGEMesaSwapBuffers` and the buffers corresponding with the sequence number of the rendering threads are tunneled.

SGEMesaQueueGet (SGEMesaContext context)*Parameters:*

context – The `SGEMesaContext` of interest.

Returns:

void* – The data item from the queue, or NULL if there is no data item.

Description:

`SGEMesaQueueGet` allows a rendering thread to retrieve information from the queue associated with the rendering thread's sequence number. If no data is available in the queue, NULL is returned. Note that a rendering thread's control loop should check for a NULL return value. If a NULL is encountered, the rendering thread should then skip its rendering steps, call `SGEPostThread`, and then try calling `SGEMesaQueueGet` again.

SGE/Mesa Reporting Functions

These functions return values maintained by the `SGEMesaContext`. Some of these functions are thread-specific and can only be run from rendering threads where noted.

SGEMesaGetSeqNo (SGEMesaContext context)*Parameters:*

context – The `SGEMesaContext` of interest.

Returns:

GLint – The sequence number which had been assigned to the calling rendering thread when it was created with `SGEMesaCreateThread`. The value of -1 is returned if there is an error.

Description:

`SGEMesaGetSeqNo` can be used to retrieve the sequence number which corresponds with the calling thread. This function only works if a rendering panel had been associated with the calling thread, either if the calling thread was created with `SGEMesaCreateThread` or if a default rendering context was created by `SGEMesaMakeCurrent`.

SGEMesaGetNodeNo (SGEMesaContext context)*Parameters:*

context – The `SGEMesaContext` of interest.

Returns:

GLint – The MPI rank number of the node on which this function is being called.

Description:

`SGEMesaGetNodeNo` can be used to retrieve the node number which corresponds with the calling node. Any thread can call this function.

SGEMesaGetTotalNodes (SGEMesaContext context)

Parameters:

context – The SGEMesaContext of interest.

Returns:

GLint – The total number of nodes in the group utilized by SGE/Mesa.

Description:

SGEMesaGetTotalNodes returns the total number of nodes in the group utilized by the SGE/Mesa tunnel. Any thread can call this function.

SGEMesaGetThreadNo (SGEMesaContext context)*Parameters:*

context – The SGEMesaContext of interest.

Returns:

GLint – The number which had been assigned to the calling thread. The value of –1 is returned if there is no number which had been assigned or if there is an error.

Description:

SGEMesaGetThreadNo returns the thread number which had been assigned to the calling thread when SGEMesaCreateThread was called. Only rendering threads can call this function.

SGEMesaGetThreadsLocal (SGEMesaContext context)*Parameters:*

context – The SGEMesaContext of interest.

Returns:

GLint – The number of rendering threads registered with the given context which exist on the calling node. The value of –1 is returned if there is an error.

Description:

SGEMesaGetThreadsLocal returns the number of rendering threads belonging to the given context on the calling node. The number of rendering threads on other nodes is not considered in this function. The returned value is not accurate until SGEMesaMakeCurrent is called after threads are created with SGEMesaCreateThread. The main thread or a rendering thread can call this function.

SGEMesaGetThreadsGlobal (SGEMesaContext context)*Parameters:*

context – The SGEMesaContext of interest.

Returns:

GLint – The total number of rendering threads registered with the given context across all nodes. The value of –1 is returned if there is an error.

Description:

SGEMesaGetThreadsGlobal returns the total number of rendering threads registered with the given context across all nodes. The returned value is not accurate until SGEMesaMakeCurrent is called after threads are created with SGEMesaCreateThread. The main thread or a rendering thread can call this function.

SGEMesaGetNumSeqs (SGEMesaContext context)*Parameters:*

context – The SGEMesaContext of interest.

Returns:

The number of unique sequence values in use among all nodes.

Description:

The value returned by SGEMesaGetNumSeqs represents the number of potentially simultaneously-rendering graphics frames (not considering multiple regions of each frame). This also is the total number of back-queue positions maintained by SGEMesaQueueStore. Avoid calling this function repetitively if possible. While this function is not slow, it is not trivial as it iterates through an array counting unique sequence values.

SGEMesaGetDimensions (SGEMesaContext context, GLint *width, GLint *height, GLint *x, GLint *y, GLint *totalWidth, GLint *totalHeight)

Parameters:

context – The SGEMesaContext of interest.

width – a return for the width of the current thread's region width

height – a return for the height of the current thread's region height

x – a return for the x-position of the current thread's region starting point

y – a return for the y-position (from the bottom of the window) of the current thread's region starting point

totalWidth – a return for the width of the total buffer (window size)

totalHeight – a return for the height of the total buffer (window size)

Returns:

GLint – 0 if there was an error or if there is no context associated with the current thread.

Description:

SGEMesaGetDimensions returns information on the size and position of the region in which the calling thread renders to. The coordinates (0, 0) for x and y correspond with the lower left-hand corner of the screen. The values returned by this function (for the region width, height, and coordinates) should be the same as what glGet operations for querying the viewport would return. Any thread can call this function which renders.

SGE/Mesa GLUT Functions

These are functions which have been added to the GLUT library to support the SGE and the parallel environment.

glutSGEGetContext ()

Returns:

SGEMesaContext – The current context utilized by GLUT.

Description:

glutSGEGetContext is provided to allow the current context maintained by GLUT to be exposed so that calls to functions such as SGEMesaCreateThread can be made.

No tests have been done to verify that this function works correctly when GLUT is managing more than one window.

glutSGESetMPIGroup (MPI_Comm group)

Parameters:

group – The group which will be used for SGE/Mesa intercommunication and SGE tunneling operations.

Description:

glutSGESetMPIGroup allows a group to be defined for use in SGE/Mesa and SGE transfers. This function must be called before glutInit is called. If this function is not utilized, then glutInit automatically calls MPI_Init and then uses the MPI_COMM_WORLD group.

glutSGESetStereo (int flag)*Parameters:*

flag – Set flag to nonzero to enable SGE stereo contexts. GLUT has its internal variable set to zero by default to disable stereo.

Description:

glutSGESetStereo enables stereo support in GLUT. Specifically, it tells GLUT to set the stereo flag when calling SGEMesaCreateContext. To enable stereo, call glutSGESetStereo before calling glutCreateWindow.

glutSGESetAlpha (int flag)*Parameters:*

flag – Set flag to nonzero to enable alpha channel support for compositing. This is zero by default.

Description:

glutSGESetAlpha tells GLUT to set the alpha flag when it calls SGEMesaCreateContext. This flag is currently not used by SGEMesaCreateContext, but could be enabled if needed.

glutSGESetDepth (int flag)*Parameters:*

flag – Set flag to nonzero to enable depth buffer support for compositing. This is zero by default.

Description:

glutSGESetDepth tells GLUT to set the depth buffer flag when it calls SGEMesaCreateContext. This flag is currently not used by SGEMesaCreateContext.

Bugs/Improvements

Currently, no alpha channel is requested when “gl_create_visual” is called in “SGEMesaCreateContext”. The driver does not currently have code to maintain an alpha channel.

The SGE/Mesa driver maintains 16-bit color data in the SGE 16-bit color format. No support for 8-bit color channels is written in this version for the SGE 32-bit color format. Note that color data read back from the buffer (either with glReadPixels or for blending) is promoted from its 5-bit width per channel to 8 bits.

GLUT text doesn't plot if the origin of the text is outside of the viewport. This means that if output is automatically divided among multiple nodes, some text will be cut off. A way to fix this is to change GLUT so that it will start text outside of the viewport, or to plot text onto an offscreen texture buffer and then render the texture.

As mentioned above, SGE/GLUT will not initialize MPI unless the "glutInit" call is made. Some demos do not call "glutInit" (but "glutInit" may be added to such code and recompiled to fix this problem).

Some demos (such as those in the "Mesa-3.4.2/book" directory) do not call any swapping functions. SGE/Mesa will not initiate a tunneling operation unless an explicit swap is made as mentioned above.

Some demos, such as the "demos/ray" or "demos/tunnel" (no correlation with SGE Tunneling) do not work because of a hack done to divide projection coordinates among nodes. Specifically, projection matrices other than the identity matrix may not be modified by glFrustum, glOrtho, or gluPerspective without problems occurring. In addition, the action of pushing or popping a projection matrix will cause the projection matrix to change unexpectedly (which is what happens in "demos/ray" and "demos/tunnel".) A current workaround for this is to always load an identity matrix and then call glFrustum, glOrtho, or gluPerspective. (A generalized "Frustum" and "Ortho" driver call in the Mesa "matrix.c" file would solve this problem easily).

No support exists for clean rendering thread destruction.

No support currently exists for canceling items submitted to the queue or for canceling completed renderings submitted for tunneling.

Miscellaneous Notes

Currently, SGE/Mesa does not perform compositing. If some simple compositing functionality is needed, it would be possible to perform compositing in the "SGEMesaSwapBuffers" function in src/sgemesa.c. The call to "sge_update" is made when a buffer is ready to be tunneled. A nicer compositor could utilize the rendering thread, freeing up the main thread; in this case, if multiple rendering threads are used, the compositor could exist in the "SGEMesaThreadPost" function before the "READY" value is assigned to the thread table. Improved SGE performance is observed when graphics data arrives from multiple nodes; it is therefore advised that the compositor does not send all of the composited graphics data to one node, but instead composites in a distributed fashion according to the established regions (or some new distribution which would require modifications to the "sge_update" function).

The "SGEMesaSwapBuffers" function currently is required to be called from the main thread the same thread which is in charge of receiving events in SGE/GLUT. "SGEMesaSwapBuffers" blocks while the SGE tunneling operation takes place, even if separate rendering threads perform rendering. It may be possible to have another thread run which performs the call to the tunnel operation as long as a mechanism is in place to prevent the SGE/GLUT calls to the SGE event handler from occurring while a tunneling operation is taking place.

Currently SGE/Mesa automatically assigns output regions as equal screen areas proportionate to the number of nodes. To change this behavior, code in the “allocBuffers” function in `sgemesa.c` can be modified. The portion of code which assigns region values is identified by a remark preceding the assignments to the “region” structure.

Code in “`sgemesa.c`” allows auto-partitioning of up to 20 nodes. To add support for more nodes, add one or more entries to the “switch” block at the start of the “allocBuffers” function. It is conceivable that one would want to try to run 64 instances of SGE/Mesa on four 16-processor nodes with the SGE GBE libraries. (Let me know how it goes!) Also note that if it is intended to try over 256 threads (or processes), increase the value in the “MAXTHREADS_SGE” definition at the start of “`sgemesa.c`”. This value controls the size of the table of regions shared across all nodes.