IEGD Linux Kernel Module
Porting and Patching Methods

November, 2009
Executive Summary

Intel® Embedded Graphics Drivers (IEGD) provide a high performance, embedded-focused graphic solution for Intel integrated graphics. It supports a wide variety of operating systems including Linux*.

The IEGD Kernel Module (IKM) is the Linux kernel component responsible for allowing the X-Window System driver access to the Intel graphics device. Its role is critical in enabling everything from simple 2D graphics and windowing subsystems to full-featured OpenGL*-based 3D applications.

Currently, IEGD supports a wide range of graphics chipsets, X versions and Linux distributions. Considering the wide range of Linux flavors available today and the propensity of embedded developers to create their own custom solutions (using custom Linux kernels), there is a need to port the IKM to these solutions.

The purpose of this white paper is to describe two methods for enabling IKM support into custom distributions and Linux kernels. First, a module-based solution is examined to port the loadable module into a newer kernel. The second solution takes this process a step further by presenting a method for creating a kernel patch so that the IKM is no longer a module, but a built-in part of the kernel.

The goal of this paper is to give the reader a general idea about the steps necessary to get the IKM working for their kernel. Additionally, by examining the details of the IKM (as a module) and the idea of a kernel patch, the reader should have a better understanding of what packaging and/or installation solution works best for them and how to employ it.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glossary</td>
<td>4</td>
</tr>
<tr>
<td>Background</td>
<td>5</td>
</tr>
<tr>
<td>IEGD Kernel Module</td>
<td>5</td>
</tr>
<tr>
<td>IEGD Kernel Patch</td>
<td>5</td>
</tr>
<tr>
<td>Solution</td>
<td>5</td>
</tr>
<tr>
<td>IEGD Kernel Module</td>
<td>5</td>
</tr>
<tr>
<td>IEGD Kernel Patch</td>
<td>5</td>
</tr>
<tr>
<td>IEGD Kernel Module</td>
<td>6</td>
</tr>
<tr>
<td>Source Structure Overview</td>
<td>6</td>
</tr>
<tr>
<td>Building the IKM</td>
<td>6</td>
</tr>
<tr>
<td>Porting Flow</td>
<td>8</td>
</tr>
<tr>
<td>Porting Areas</td>
<td>9</td>
</tr>
<tr>
<td>Kernel Headers</td>
<td>9</td>
</tr>
<tr>
<td>Kernel Dependency Checker</td>
<td>9</td>
</tr>
<tr>
<td>Build Failures</td>
<td>10</td>
</tr>
<tr>
<td>AGP and DRM Considerations</td>
<td>10</td>
</tr>
<tr>
<td>X Considerations</td>
<td>11</td>
</tr>
<tr>
<td>IEGD Kernel Patch</td>
<td>11</td>
</tr>
<tr>
<td>Creating a Kernel Patch</td>
<td>11</td>
</tr>
<tr>
<td>Copying Source Files</td>
<td>11</td>
</tr>
<tr>
<td>Creating and Updating the Makefiles</td>
<td>12</td>
</tr>
<tr>
<td>Updating Kconfig</td>
<td>13</td>
</tr>
<tr>
<td>Additional Requirements</td>
<td>13</td>
</tr>
<tr>
<td>Creating the Patch</td>
<td>13</td>
</tr>
<tr>
<td>Applying a Kernel Patch</td>
<td>13</td>
</tr>
<tr>
<td>Conclusion</td>
<td>14</td>
</tr>
</tbody>
</table>
## Glossary

Glossary of terms.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGP</td>
<td>Accelerated Graphics Port. This also refers to the infrastructure in Linux responsible for resource management of graphics devices (&quot;AGP&quot; term has historical connotations despite recent graphics cards being PCIE-based).</td>
</tr>
<tr>
<td>DRM</td>
<td>Direct Rendering Manager. This component is implemented in the kernel and is primarily used by the X-Window System to access graphics hardware in a safe and efficient manner.</td>
</tr>
<tr>
<td>HOST</td>
<td>Development and build system. This is where kernel patches or the IKM is built.</td>
</tr>
<tr>
<td>IEGD</td>
<td>Intel Embedded Graphics Driver</td>
</tr>
<tr>
<td>IKM</td>
<td>IEGD Kernel Module. This is the component which integrates with the Linux kernel to allow the X-Window driver to access the graphics device.</td>
</tr>
<tr>
<td>PATCH</td>
<td>Kernel patch. This is a file which contains the deltas that are merged into the kernel source.</td>
</tr>
<tr>
<td>TARGET</td>
<td>System running the HOST build binaries</td>
</tr>
<tr>
<td>X-Window System</td>
<td>This is the Graphical User Interface (GUI) in Linux (and many other operating systems).</td>
</tr>
</tbody>
</table>
Background

**IEGD Kernel Module**

The IEGD Kernel Module (IKM) is the Linux kernel module responsible for allowing the X-Server driver to allocate and use the graphics device resource (for example, kernel pages for video operations). The IKM supports several Linux distributions and more specifically, several kernel versions. With as many kernel versions that exist and with the need to support future kernels, the IKM is in a constant state of being updated. Although the IKM currently supports many kernel versions, there are unsupported distributions and kernels that customers may want to use.

**IEGD Kernel Patch**

The IEGD kernel patch is a way to compile the IKM code into the kernel itself, instead of being a separate module. With the module approach, the process to install IEGD onto a Linux platform is to compile and install the IKM, insert the kernel module into the kernel, and then copy the required drivers to the installation. An alternative is to have IEGD pre-installed so there is no IEGD setup or installation. Moblin is an example of where this is used.

Solution

**IEGD Kernel Module**

To support other distributions or kernel versions, the IKM must be ported to the newer kernel version. At a high level, this mainly involves ensuring that new kernel dependencies are resolved so that the IKM builds and properly integrates itself with the graphics infrastructure. This can be a simple task such as updating to the latest set of header files, or it can be something much more complicated, depending on the specific parts of the kernel that differ from what the current IKM supports. Regardless of the situation, the solution to get the IKM working on an unsupported distribution and/or kernel is to port the IKM. That process is what is described here.

**IEGD Kernel Patch**

For pre-packaging IEGD into a distribution such as Moblin, it is much easier and straightforward to have the IEGD kernel module compiled directly into the kernel rather than inserting a separate kernel module. A natural way to add IEGD into a kernel is to create an IEGD kernel patch file. When the patch is applied, it will add the IEGD code to the kernel source. When the kernel is compiled, the IEGD code will be included in the kernel itself. With the IEGD included in the kernel and by packaging the needed drivers in the distribution, IEGD works out-of-the-box with no installation.
IEGD Kernel Module

This section describes the process of porting the IEGD Kernel Module to an unsupported kernel. First, the structure of the IKM source and how to build the IKM are discussed. Then porting consideration and areas for potential modification are reviewed.

Source Structure Overview

The IKM’s current source tree is illustrated below.

- **The IKM directory** is the root of the IKM source.
- **The agp directory** contains the code for initializing the iegd_mod.ko driver module and for integrating the module into the kernel’s AGP infrastructure. This infrastructure is responsible for calling back into the driver for all resource management tasks requested by the kernel and X-Server driver.
- **The drm directory** contains the code that allows the module to integrate with the kernel’s DRM (Direct Rendering Manager) subsystem. This component is primarily used by the X-Window System to access graphics hardware in a safe and efficient manner.
- **The kernelchecker_tests directory** contains several tests that are used by the IKM/install.sh script to ensure that kernel APIs and structure dependencies exist in the kernel for which the IKM is being compiled and to flag any potential IKM build issues.
- **The val directory** contains abstractions and include files for specific AGP and DRM versions. When either of these components gets updated in a significant manner that impacts the IKM, a new val/* directory is created so that the new kernel-dependent files are included for the new kernel.

Building the IKM

Building the IKM is typically handled by running the IKM/install.sh command. This command builds the IKM for the current kernel-booted kernel. If another kernel is desired, its headers need to be pointed to by the KERNELDIR environment variable. This variable should be set prior to executing IKM/install.sh.
Building the IKM does the following:

1. Identifies the current kernel, that is, the one for which kernel the IKM will build.

2. Selects the correct IKM/val/* directories so that the correct AGP and DRM headers are used for the current IKM build.

3. Checks Kernel Dependency which determines the kernel headers by building various test modules to ensure the IKM can be built for that particular kernel.

4. Generates a Makefile specific to the kernel headers.

5. Builds the IKM.

6. Copies the IKM to the kernel module directory and installs the module.

7. Executes the AGP and DRM tests to ensure that the install was a success.

After the installation is complete, the IKM module is available to the X-Server driver.
Porting Flow

The flow chart below describes the typical flow when porting IKM to a new kernel.

1. Run ./install.sh
2. Check if the kernel headers are found. If not, install the kernel development headers for the target to build the system. Alternatively, set KERNELDIR to point to the headers.
3. Analyze build failures.
4. Install a dependency check if the build fails.
5. IKM compiled and linked successfully?
   - Yes: Analyze and fix compile/link errors.
   - No: Debug AGP; Examine /var/log/messages for kernel failures.
6. Agp_test executed successfully?
   - Yes: Debug DRM; Examine /var/log/messages for kernel failures.
   - No: Copy X driver, port, DRI and xorg.conf files per User Guide.
7. Check glxinfo for direct rendering enabled?
   - Yes: Installed OpenGL?
     - Yes: Got display?
       - Yes: Run ./install.sh
         - Done
       - No: Start xserver.
     - No: Examine /var/log/Xorg.0.log
   - No: Enable DRM.

Done
Note: For the porting, Intel recommends starting with a known-good xorg.conf file that has already been validated on the target hardware platform using a natively supported Linux distribution. This will remove the complication of troubleshooting xorg.conf and panel timing issues in combination with IKM porting changes.

Porting Areas

Typically, porting an IKM to a new distribution or kernel involves trying to run ./install.sh and seeing what parts of the installation breaks. As can be seen from the flow chart, failures can happen in several places including the kernel dependency checker, the compile/build of the IKM, errors in the integration of the AGP or DRM and executing X. The following sections describe some of the areas of consideration during the port process.

Kernel Headers

The install.sh builds the IKM using the kernel headers that it finds. It does this by examining the boot kernel version and using that version information to find the analogous kernel headers. For platforms where the IKM is being built on the HOST and the TARGET is the same as the HOST, this method works fine. For systems where the HOST is being used to build for a different TARGET, a different set of headers may need to be specified. To accommodate this, the install.sh recognizes the KERNELDIR environment variable. This variable can be set to point to the TARGET’s kernel header files. These headers will then be used during the IKM build process.

Note: Typically, installing the kernel headers can be accomplished at the time the Linux distribution is installed or by installing the “kernel-devel” package at a later time (via the source packages, Yum, Synaptic, etc.).

Kernel Dependency Checker

The purpose of the kernel dependency checker script is to provide a mechanism that determines compile time failures before compiling the IKM. The kernel dependency checker does this by building many small kernel modules that utilize kernel structures and programming interfaces, that is, the kernel structures and programming interfaces used by the IKM. Because these modules (or tests) are compiled, warnings and errors can be used to debug potential IKM build issues.
Build Failures

The kernel dependency checker is meant to cover most of the kernel structures and programming interfaces used by the IKM, but considering the build is for a new or unsupported kernel, the dependency checker may not catch all compiler issues. In this case, the kernel dependency checker allows the IKM build to happen, but this may result in compiler or linker errors. In this case, each error must be resolved to successfully build the IKM. The errors will generally be in the following categories:

- Programming interface change
- Structure changes
- New interfaces or infrastructure design changes

Programming interface changes can be as simple as name changes, or something more involved such as return value changes, parameter list changes, or usage changes. Structure changes are similar. These changes are usually changes to the structure member name, additional members, or removed members.

The last category is a bit ambiguous. As the kernel progresses, changes to the kernel's architecture and sub-architectures can happen. The IKM relies on several of the kernel's components for memory management, synchronization, interrupts, and other kernel objects. Changes in any of these components (and their associated structures) can affect the way that the IKM must work.

AGP and DRM Considerations

The IKM depends on the AGP and DRM to "plug itself" into the kernel's AGP graphics resource management and direct rendering infrastructures. Both of these components have existed for some time. When considering build failures in these areas, it's important to consider the dependencies that the IKM has for these areas. From a programmatic perspective, these dependencies are exposed via the kernel's AGP and DRM header files.

For the IKM, the AGP and DRM header files located in the IKM/val/agp and IKM/val/drm directories are used during the compile, depending on the kernel version for which the IKM is being compiled. To determine which AGP and DRM directory is being used, please note that after the ./install.sh executes, it creates a Makefile.gnu which is used to make the IKM. This is a generated makefile that is specific for the kernel version being compiled. In the case of a new kernel, the ./install.sh will do its best to choose the best set of headers to use.

Note: Incorrect AGP and DRM headers may allow the IKM to compile correctly, but will most likely result in kernel panics or incorrectly working IKM.

To deal with this situation, it's imperative that the AGP and DRM headers in the kernel exactly match the headers used by the IKM in the val subdirectory. This is simply a matter of getting the kernel source and copying the AGP and DRM headers to the correct subdirectories within IKM/val/agp/ and IKM/val/drm/. Use the generated Makefile as a guide for which header files will be needed for the compile and which val directory they need to be in. Kernel source directories drivers/char/agp and include/drm are good places to begin to find the headers.
Alternatively, new `val/agp/myagp` and `val/drm/mydrm` directories can be created to satisfy a specific new build. To do this, adjustments to the `install.sh` and/or `makefile` will be required, so that these directories are the ones that are used during the compile.

**Note:** For all kernel failures, examining `/var/log/messages` is the best way to check for IKM-, AGP-, and DRM-related issues.

**X Considerations**

First, ensure that a valid `xorg.conf` is in place and that the IEGD driver files have been copied to the correct destinations (per the IEGD User’s Guide). Make any changes specific to the TARGET’s distribution and then start X, which is the ultimate goal when getting the IKM ported to a new kernel. At this point, assuming the IKM is installed, AGP test passed, and DRM test passed, X should work without issue. If issues arise, the best place to start debugging is to look at `/var/log/Xorg.0.log`. This file contains a running log of X-related issues and errors, including any IEGD driver issues.

**IEGD Kernel Patch**

This section describes the steps required to create a kernel patch from the IEGD Kernel Module. Please note that the IKM must fully work on the kernel for the kernel patch to work. If the IKM does not work, please refer to the IEGD Kernel Module section on porting the IKM to a new kernel (see page 9).

**Creating a Kernel Patch**

To prepare for creating a kernel patch, the full kernel source and IKM source are needed. The kernel source must be the exact kernel version which will be running. For illustration purposes in this document, the example name for the original full kernel source directory will be `vanilla_kernel`.

**Copying Source Files**

Duplicate `vanilla_kernel` and rename it to `target_kernel`. The duplicate kernel `target_kernel` should be in the same directory as `vanilla_kernel`. The kernel is duplicated so that the original kernel source, which is needed to create the patch, will be left untouched.

Create a new directory called `iegd` in the directory `target_kernel/drivers/gpu/drm/`. If the kernel is older than 2.6.27, the location will be `target_kernel/drivers/char/drm/`.
From IKM directory, copy agp/, drm/, include/ directories to the iegd/ subdirectory in target_kernel. Those three directories have the core source code of the IKM. Next, from IKM/val directory of the specific distribution, look for the file interface_abs.h. For the Moblin distribution, it is located at IKM/val/agp/agpm0vmob2/interface_abs.h. Copy this file into the iegd/include directory within the target kernel. This file is used for interfacing between the core source code of the IKM and the kernel for the particular distribution.

Creating and Updating the Makefiles

An iegd makefile must be created for the kernel. Create an empty file called Makefile in target_kernel/drivers/gpu/drm/iegd/ and fill it with the following code:

```bash
ccflags-y := -Idrivers/gpu/drm/iegd/include
ccflags-y += -Idrivers/char/agp -Iinclude/drm
iegd_mod-objs := agp/pci.o agp/global.o agp/drv alm.o agp/drv nap.o agp/drv plb.o agp/drv cmn.o agp/drv gn4.o drm/iegd drv.o drm/iegd interface.o drm/iegd interface 265.o drm/iegd interface 2611.o drm/iegd interface 2615.o drm/iegd interface 2624.o drm/psb_irq.o
obj-y += iegd_mod.o
```

This makefile code is based on the IKM Makefile, which is generated by install.sh in the IKM directory. However, please note the differences in the files. The ccflag include path is defined with respect to the kernel source tree root, instead of being the full disk directory paths. The ccflag include paths do not point to the IKM val directories, but rather point to agp and drm headers within the kernel source tree. The iegd_mod-objs code is exactly the same in the IKM Makefile and the iegd kernel makefile. The obj-y code has an important difference. The ‘y’ sets the compiler to include the module in the kernel, so the ‘y’ is essential. The IKM Makefile has obj-m, which makes it compile as a separate module. Finally, please note that if the kernel is older than 2.6.27, the ccflags path for iegd/include will be incorrect. In the path, change the gpu to char.

Update the existing makefile in drm/ to include the iegd/ module. Open for editing the file target_kernel/drivers/gpu/drm/Makefile. At the end of the file, append the line

```bash
obj-$(CONFIG_IEGD) += iegd/
```

This will include the iegd module into the kernel based on the CONFIG_IEGD parameter.
Updating Kconfig

Next, update Kconfig to handle the new CONFIG_IEGD parameter. Kconfig is used in the Linux kernel to list all the modules and whether they should be included in the kernel, ignored, or compiled as a separate module. Open for editing the file target_kernel/drivers/gpu/drm/Kconfig. At the end of the file, append the lines:

```c
config IEGD
tristate "Intel IEGD"
depends on DRM
help
Choose this option for the Intel Embedded Graphics Driver (IEGD)
```

Note: In the Kconfig file the naming of the config is IEGD. This will be matched up with the CONFIG_IEGD parameter in the makefile. Before the kernel can be compiled with the command make, the .config will be updated (via make oldconfig or similar commands) and the user will be given a chance to include the iegd module in the kernel.

Additional Requirements

In target_kernel, if include/linux/config.h does not exist, create the file. The file can be blank, but it needs at least one character (such as a space) for the patch command to include it. Code in the IKM includes the config.h file, and the compile will fail if the file does not exist. Simply creating a file with that name allows the compile to pass.

Creating the Patch

Run the patch command to create kernel patch. Change the working directory to the directory that has both vanilla_kernel and target_kernel. The command is:

```bash
diff -uNr vanilla_kernel target_kernel > file.patch
```

Applying a Kernel Patch

To apply the patch to a kernel source tree, copy the patch to the parent directory of the kernel. Change the working directory to the kernel source tree root. Then use the following command:

```bash
patch -p1 < ../file.patch
```
Conclusion

Two methods were described for enabling IKM support into custom distributions and Linux kernels. The first method was a module-based solution. The process was explained to port the loadable module into an unsupported kernel and where problems may arise. The second method was to take this process a step further by presenting a method for creating a kernel patch so that the IKM is no longer a module, but a built-in part of the kernel.

The reader should now have a general idea about the steps necessary to get the IKM working for their kernel. Additionally, by examining the details of the IKM (as a module) and the idea of a kernel patch, the reader should now have a better understanding of what packaging and/or installation solution works best for them and how to employ it.