SANDIA REPORT

SAND2006-6828
Unlimited Release
Printed November 2006

SummitView 1.0: A Code to Automatically Generate 3D Solid Models of Surface Micro-machining based MEMS Designs

Corey L. McBride, Victor Yarberry, Ray Meyers, and Rodney Schmidt

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2006-6828
Unlimited Release
Printed November 2006

SummitView 1.0: A Code to Automatically Generate 3D Solid Models of Surface Micro-machining based MEMS Designs

Corey L. McBride and Ray Meyers
Elemental Technologies
17 North Merchant Street
American Fork, UT 84003

Victor Yarberry and Rodney Schmidt
Sandia National Laboratories
P.O. Box 5800
Alburquerque, NM 87185

Abstract

This report describes the SummitView 1.0 computer code developed at Sandia National Laboratories. SummitView is designed to generate a 3D solid model, amenable to visualization and meshing, that represents the end state of a microsystem fabrication process such as the SUMMiT (Sandia Ultra-Planar Multilevel MEMS Technology) V process. Functionally, SummitView performs essentially the same computational task as an earlier code called the 3D Geometry modeler [1]. However, because SummitView is based on 2D instead of 3D data structures and operations, it has significant speed and robustness advantages. As input it requires a definition of both the process itself and the collection of individual 2D masks created by the designer and associated with each of the process steps. The definition of the process is contained in a special process definition file [2] and the 2D masks are contained in MEM format files [3]. The code is written in C++ and consists of a set of classes and routines. The classes represent the geometric data and the SUMMiT V process steps. Classes are provided for the following process steps: Planar Deposition, Planar Etch, Conformal Deposition, Dry Etch, Wet Etch and Release Etch.

SummitView is built upon the 2D Boolean library GBL-2D [4], and thus contains all of that library's functionality.

Contents

Appendix

List of Figures

Chapter 1

Introduction and Background

The development of advanced multi-level surface micro-machining (SMM) micro-fabrication technologies such SUMMiT V [5] has enabled the ability to create very complex 3D MEMS devices. The 3D structure of a MEMS device results from the interaction of the individual fabrication process steps and the associated set of 2D layout masks created by the designer. However, because the 2D masks do not directly reveal 3D geometry, the structures that result from this interaction can be very difficult to visualize without special tools. In addition, the development of a finite-element mesh suitable for accurately representing the geometric complexities of the design for needed analysis is likewise problematic.

To help address these problems, Sandia has previously developed several computational design tools [6]: the 2D Process Visualizer [7, 8], a Design Rule Checking System[9], and a first generation 3D Geometry Modeler [1]. The 3D Geometry Modeler is a solid geometry modeler based on the ACIS kernel [10]. Its function is to simulate the interaction between the SUMMiT V process and a designer's mask set, at the geometric level, to create a 3D model suitable for visualization, rapid prototyping, and analysis.

The usefulness of the 3D Geometry Modeler as a design tool at Sandia prompted a subsequent effort to improve upon it. The main limitations of this first generation tool are: low robustness, large computational costs, and difficulty producing numerical meshes of the final 3D models.

The limitations with robustness and computational cost are directly linked to the use of the ACIS 3D Geometric Modeling Engine for data structures and geometric routines. ACIS is designed as an extremely general tool that can be used as:

> "The geometry foundation within virtually any end user 3D modeling application" [10].

This broad scope requires the data structures and routines to be sophisticated and flexible enough to handle a wide variety of modeling situations. However, this sophistication and flexibility comes at the price of increased size and complexity. Extensive experience with the 3D Geometry Modeler has made it clear that the use of ACIS data structures and routines to perform numerous large and complicated 3D Boolean operations is in large measure responsible for the low robustness and large computational costs of the 3D Modeler.

The difficulty in producing numerical meshes comes from the complicated 3D nature of the final

ACIS models. The 3D Geometry Modeler simulates each step of the SUMMiT V process by manipulating the model geometry to produce results consistent with the actual fabrication steps. Each process step is simulated in order, with the model that results from the current step being used as input for the next step. Thus the final geometry of ACIS model evolves as each process step is simulated. Intricate MEMS models that are produced using this development method are more complicated than those produced by modeling the geometry directly. It is this added complexity that is largely responsible for the difficulty in producing numerical meshes.

The SummitView computer code described here represents the results of a concerted effort to overcome the aforementioned problems of the 3D Geometry Modeler. A key aspect of the SummitView code is that all geometric objects are what can be called "$2\frac{1}{2}$D." That is, they consist of a 2D geometry with an associated thickness. Using $2\frac{1}{2}$ D geometry allows the modeling problem to be completely solved using 2D data structures and 2D geometric routines. Functionally, SummitView performs essentially the same role as the 3D Geometry modeler. However, because it is based on 2D instead of 3D data structures and operations, it has significant speed and robustness advantages. In addition, it can be far simpler to produce numerical meshes of $2\frac{1}{2}$D geometries than of 3D geometries

To perform the various 2D Boolean operations that represent actions occurring during different process steps, SummitView relies on a custom 2D geometric Boolean library, called GBL-2D [4], that was developed in parallel with SummitView. Developing a custom modeling engine as part of this work had many advantages. For example, by freeing the developer from using ACIS routines, it allowed for source-level control of data structures and geometric routines. In addition to allowing greater flexibility, this also allows for data structures and routines to be stream-lined for a specific application. Stream-lined data structures and routines leads directly to reduced computational costs and improved overall robustness. These advantages have been verified by tests comparing SummitView 1.0 with the 3D Geometry modeler that demonstrated a consistent speed-up of approximately 2 orders of magnitude.

This report describes SummitView 1.0, a 2nd generation code to automatically generate 3D solid models of SUMMiT V MEMS designs.

Chapter 2

Overview of SummitView 1.0

SummitView 1.0 is written in C++ and consists of a set of C++ classes and associated routines. These classes can be divided into two categories: (1) classes that represent geometric data, and (2) classes that simulate the SUMMiT V process steps. The SummitView Library is built upon and therefore also includes all of the classes in the 2D Boolean library GBL-2D [4]. In the code, the class names in SummitView begin with the prefix "SV" to distinguish them from those classes which belong to GBL-2D, which begin with "DD".

An example of how to use these C++ classes and associated routines to produce $2\frac{1}{2}$ D geometry is found in Appendix 1. A definition of the fabrication process and the 2D masks created by the designer are required as input. The definition of the fabrication process is contained in a special process definition file (see [2]), and the 2D masks are contained in a MEM format file (see [3]).

2.1 Supported Compilers

The SUMMiTView Library has been tested with the following 32 bit compilers.

- Microsoft Visual C++ 6.0 with STLport Version 4.5.3 [11]
- Microsoft Visual C++ .NET 2003
- Microsoft Visual C++ 2005 Express Edition
- G++ 3.2 or greater

2.2 $2\frac{1}{2}$D Data classes

SummitView 1.0 models a $2\frac{1}{2}$D geometry using a collection of what are called here "chunks." A chunk consists of a 2D surface defined in the X-Y plane that is extruded in the Z direction. Each volume represented by a chunk is considered to be made of a single specific material. In the code, a chunk is represented by an instance of SVChunk.

The material used to define each SVChunk comes from an instance of SVMaterial. An instance of SVMaterial contains a root material name and a step number. There may be several instances of SVMaterial with the same root material. However, there will only be one instance of SVMaterial for each step number. Maintaining this distinction allows the individual parts of the final geometry to be analyzed by both material and by the step from which it was deposited. Under normal situations an instance of SVMaterial is created for each deposition step. ”Void” is considered a valid material and is used to define the SVChunks representing empty space.

The surface used to define a SVChunk comes from an instance of DDSurface. For the purpose of the SummitView, each instance of DDSurface belongs to a set of interconnected DDSurfaces that tile the geometry’s bounding box in the X-Y plane. Interconnected DDSurfaces share instances of DDEdge at their neighboring boundries. There is an instance of SVChunk for each DDSurface in the interconnected set at each elevation in the geometry.

Each unique elevation in the geometry is defined using an instance of SVLevel. An SVLevel contains a thickness and a list of SVChunks. The thickness is a relative distance in the Z direction from the bottom of the level to the top of the level. Each SVLevel is defined throughout the geometry. Thus, there is a SVChunk for each DDSurface at each SVLevel. The complete set of SVChunk instances title the 3D bounding volume of the MEMS Geometry. It is helpful in understanding the SummitView routines to consider the 3D bounding volume as a 3D grid. The X-Y divisions are defined by the interconnected set of DDSurfaces. The Z divisions are defined by the instances of SVLevel.

2.3 SUMMiTView Process Step Classes

This section describes the classes used to simulate the SUMMiT V process steps. Each sub-section provides a description of a class and the algorithm used to simulate the process step. Specific details of each class and it’s interface methods can be found in Chapter 4.

To better understand the algorithms used to simulate the process steps it is important to understand two concepts. The first concept is the idea of an SVChunk being reachable. A SVChunk is considered reachable if it is connected to the top of the geometry by a chain of neighboring SVChunks composed of affected material. The affected material depends upon the individual process step. For some process steps the affected material may only be ”Void” material. While for other process steps the affected material may be several materials.

The second concept to understand is the idea of neighboring SVChunks. A SVChunk is conisdered to be a neighbor to another SVChunk if the two SVChunks share the same DDSurface or are adjacent to each other. Because there is a SVChunk for each DDSurface at each SVLevel, a DDSurface represents a column of geometry that extends in the Z direction throughout the complete geometry. Thus the neighbor to a SVChunk can be found by traversing the column of SVChunks up or down one SVLevel. Or in other words, the neighbor to a SVChunk can be found be finding the SVChunk at the level above or below the current level that shares the same DDSurface.

Two SVChunks are considered to be adjacent to each other if their respective DDSurfaces share one or more instances of DDEdge at their boundary. Two DDSurfaces are not considered to be adjacent if they only share a common tangent point. Adjacent SVChunks are found by examining each DDEdge of the SVChunk's DDSurface. DDSurfaces that share the same instance of DDEdge are adjacent and thus the respective SVChunk are adjacent.

2.3.1 Planar Deposition

The purpose of this class is to simulate a uniform deposition of a specified material followed by a chemical-mechanical polishing. The uniform deposition is performed by finding all "Void" SVChunks that are reachable from the top of the geometry. The material of these reachable chunks is changed to the specified deposition material.

The chemical-mechanical polishing is performed by inserting a SVLevel at the deposition thickness. The deposition thickness is defined from the base of the geometry. The material of the SVChunks in the level above the new SVLevel is changed to "Void". Any SVLevels above this "Void" level are removed.

Figure 2.1(a) shows an existing $2\frac{1}{2}$D geometry. Figure 2.1(b) shows the result of a planar deposition with a thickness that is less than the height of the original geometry. Thus the portion of the original geometry that extends above the deposition thickness is removed.

2.3.2 Planar Etch

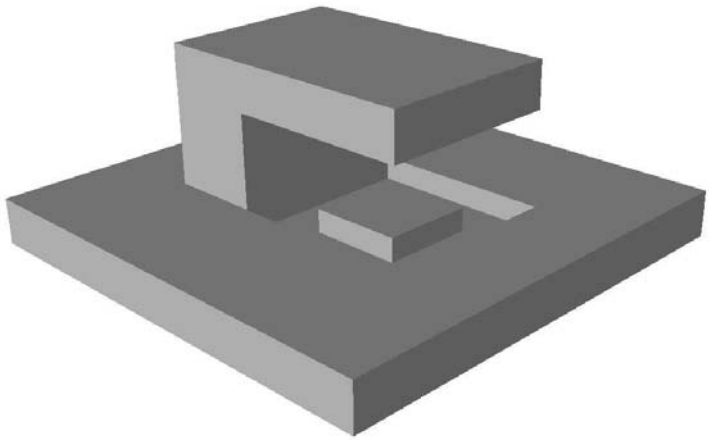
The purpose of this class is to simulate a chemical-mechanical polishing. This process produces geometry similar to a Planar Deposition without the initial deposition. The thickness of the etch is defined from the base of the geometry. Any geometry that extends above the etch thickness is removed.

The etch is performed by inserting a SVLevel at the deposition thickness. The material of all SVChunks in the SVLevel above the new SVLevel is changed to "Void". Any SVLevels above this "Void" level are removed.

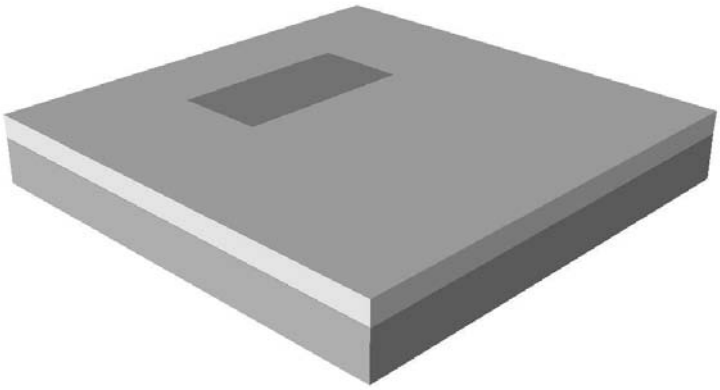
2.3.3 Conformal Deposition

The purpose of this class is to simulate a chemical vapor deposition of a specified material. The deposition is performed using a rolling ball offset in the X-Y plane and a straight offset in the Z direction. Different deposition thickness are available for the top, bottom and sides of the geometry.

The deposition is performed by first depositing material on the sides of the geometry. This is done by finding all SVChunks of "Void" material that are reachable from the top of the geometry. These



(a) Before Planar Deposition.



(b) After Planar Deposition.

Figure 2.1. Planar Deposition

chunks are grouped by level into clusters of adjacent chunks.

For each adjacent chunk cluster a DDVirtualSurface is created. This virtual surface represents the total area of all the surfaces contained in adjacent chunk cluster. The virtual surface is offset using a rolling ball offset. The offset distance is the negative side deposition thickness. This creates a surface or surfaces that are smaller than the original virtual surface. These smaller offset surfaces represents a deposition on the exposed surfaces that border the ”Void” adjacent chunks.

The offset surfaces are imprinted into original surfaces of the adjacent chunk cluster. The new surfaces that come from the difference of the offset surfaces and the original surfaces represent new chunks of the deposition material. The material of these new offset chunks are changed from ”Void” to the new deposition material.

Once all of the side deposition have been completed the depositions on the exposed top and bottom surface are done. Top an bottom depositions are similar and involves a process of examining each adjacent chunk cluster for top or bottom exposed surfaces. A top and bottom exposed surfaces is found by checking to see if any of the ”Void” chunks from the cluster border a chunk directly above or below that is not of ”Void” material.

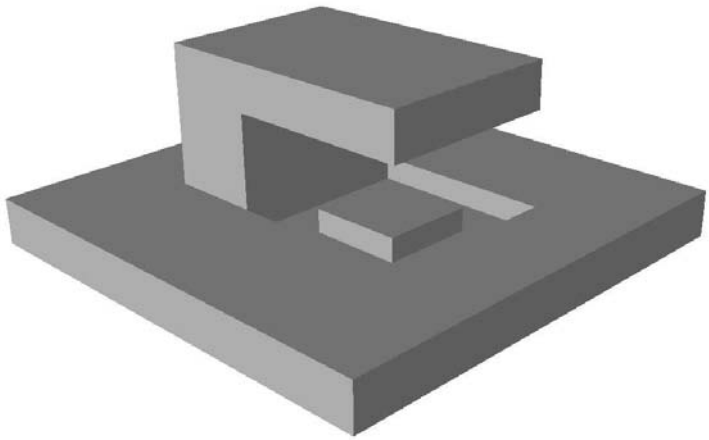
The exposed surface is deposited on by comparing the neighboring ”Void” chunks level thickness with the top or bottom deposition thickness. The neighboring ”Void” chunk is considered because the material is deposited into the neighboring void. If the deposition thickness is equal to the level thickness the material of the ”Void” chunk is changed to the deposition material. If the deposition thickness is less than the level thickness the level is split. The material of the new SVChunks that are created from the split and border the exposed surface are changed to the deposition material. If the deposition thickness is greater than the level thickness the material of the ”Void” chunk is changed to the deposition material. The remaining deposition thickness, calculated by subtracting the current level thickness from the deposition thickness, is used to continue depositing into the ”Void” chunks above or below the new exposed surface.

Figure 2.2(a) shows an existing $2\frac{1}{2}$D geometry. Figure 2.2(b) shows the result of a conformal deposition. The thickness of this example deposition in uniform on the top, bottom and sides of the exposed surfaces. The surfaces that are on border of the geometry’s bounding box are not considered exposed.

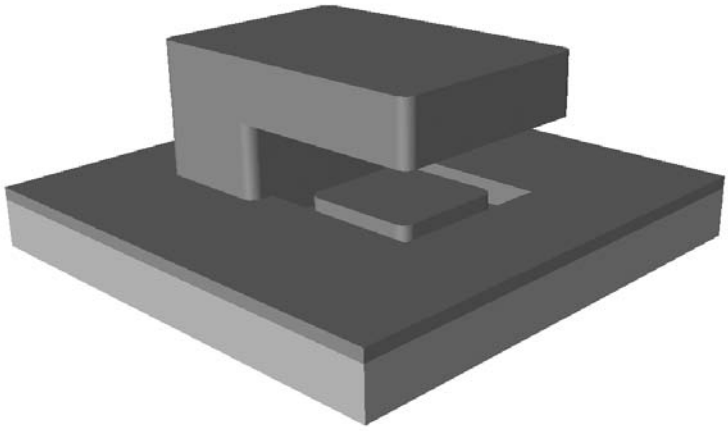
2.3.4 Dry Etch

The purpose of this class is to simulate a dry etch. The geometry of the etch is determined by a MEMS mask set. The etch affects user specified materials and is of a uniform etch depth. The $2\frac{1}{2}$D nature of the domain results in an etch that has 90 degree etch angles in the Z direction.

This etch is performed by imprinting the MEMS mask set onto the surface list of the SVMEMS-Geometry. The new surfaces that come from the intersection and difference of the mask set with the geometry surfaces are used to etch the geometry. If the mask set defines those areas that are to be etched then the intersection surfaces are used. If the mask defines those areas that are to be



(a) Before Conformal Deposition.



(b) After Conformal Deposition.

Figure 2.2. Conformal Deposition

protected from the etch then the difference surfaces are used.

Each surface is the SVMEMSGeometry surface list represents a column of SVChunks made up of ”Void” and other materials. For each intersection or difference surface, the column of related SVChunks is processed. Starting at the top of the column of SVChunks, the material of each chunk is compared to the list of affected materials. If the material of the SVChunk is affected then the thickness of the current level is compared to the remaining etch depth. At the top of the geometry the remaining etch depth is equal to the etch depth. But at the etch progresses through the geometry the remaining etch depth decreases until the etch is complete.

If the remaining etch depth is less than the level thickness the level is split at the remaining etch depth. The material of the new SVChunks created from the split are changed to ”Void”. The remaining etch depth is now zero and the etch for that column is completed.

If the level thickness is equal to the etch depth than the material of the affected chunk is changed to ”Void”. The remaining etch depth is zero and the etch for that column is complete.

If the remaining etch depth is greater than the level thickness the material of the affected chunk is changed to ”Void” The remaining etch depth is the difference of the previously remaining etch depth and the current level thickness. The new remaining etch depth is then used when examining the next SVChunk in the column. This process continues until the remaining etch depth equals zero or until a SVChunk is encountered that is not of the affected material. SVChunks of ”Void” material are ignored when performing an etch and calculating the remaining etch depth.

Figure 2.3(a) shows the surfaces that make up a mask set. Figure 2.3(b) shows the results of a dry etch. The original geometry is a block of a single material. In this case the etch depth is less than the thickness of the original geometry.

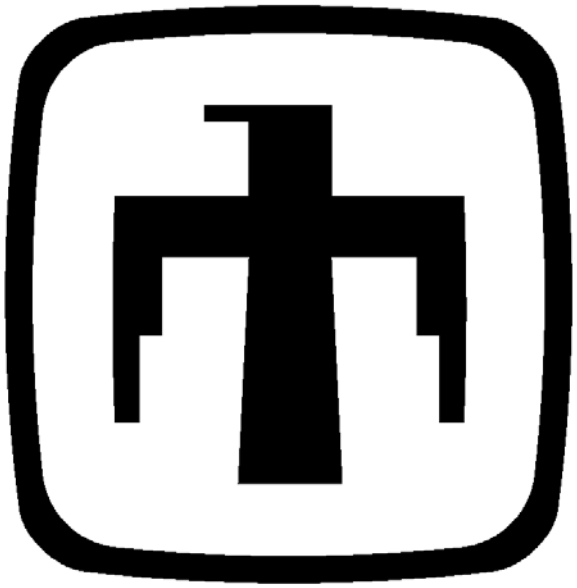
Figure 2.4(a) shows an existing $2\frac{1}{2}$D geometry consisting of three different materials. Figure 2.4(b) shows the results of a dry etch affecting one material. In this case the etch depth is greater than or equal to the thickness of the affected material.

2.3.5 Wet Etch

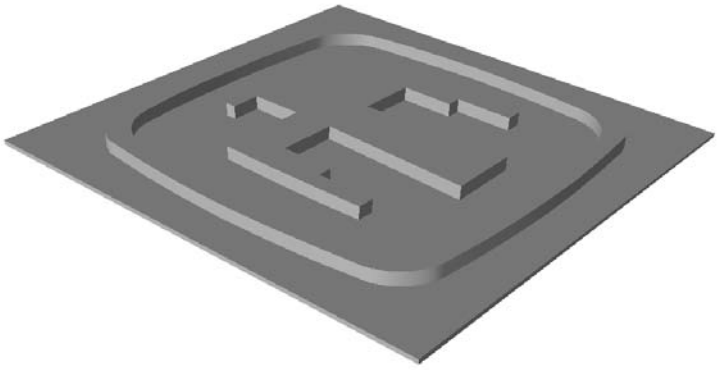
The purpose of this class is to simulate a wet etch. The etch is performed using a rolling ball offset in the X-Y plane and a straight offset in the Z direction. The etch is performed on all exposed surfaces of a user specified affected material. The $2\frac{1}{2}$D nature of the domain results in an etch that has 90 degree etch angles in the Z direction.

The etch is performed by first etching material on the sides of the geometry. This is done by finding all SVChunks of ”Void” material that are reachable from the top of the geometry. These chunks are grouped by level into clusters of adjacent chunks.

For each adjacent chunk cluster a DDVirtualSurface is created. This virtual surface represents the total area of all the surfaces contained in adjacent chunk cluster. The virtual surface is offset using

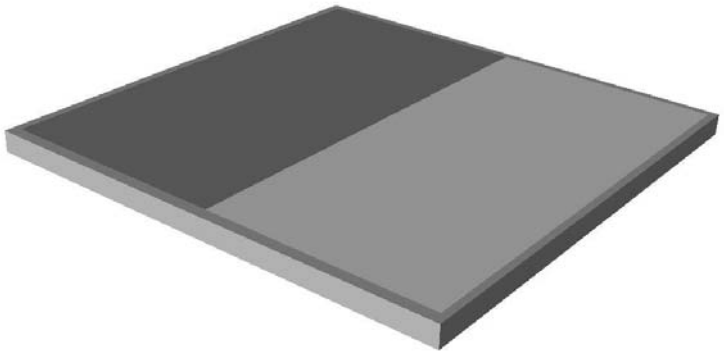


(a) Surfaces that define the etch mask.

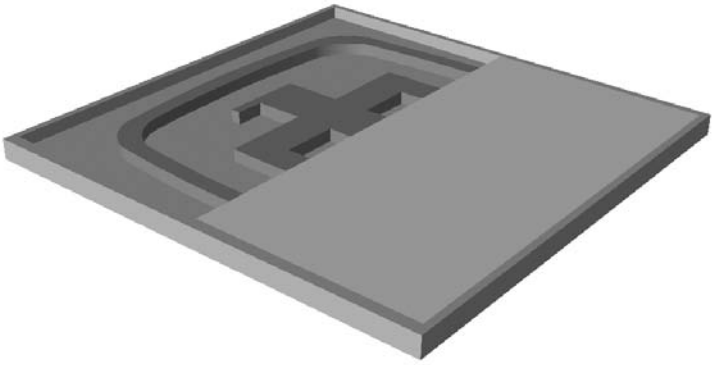


(b) Results of a dry etch.

Figure 2.3. Dry Etch



(a) Before Dry Etch.



(b) After Dry Etch.

Figure 2.4. Dry Etch of one material.

a rolling ball offset. The offset distance is the etch depth. This creates a surface or surfaces that are larger than the original virtual surface. These larger offset surfaces represents an etch of the exposed surfaces that border the ”Void” chunks.

The offset surfaces are then imprinted into original surfaces of the adjacent chunk cluster. The new surfaces that come from the intersection of the offset surfaces and the original surfaces represent the material that was etched away. The material of these new offset chunks are changed to ”Void” material.

Once all of the side etching have been completed the exposed top and bottom surface are etched. Top an bottom etchs are similar and involves a process of examining each adjacent chunk cluster for top or bottom exposed surfaces. A top and bottom exposed surfaces is found by checking to see if any of the ”Void” chunks from the cluster border a chunk directly above or below that is not of ”Void” material.

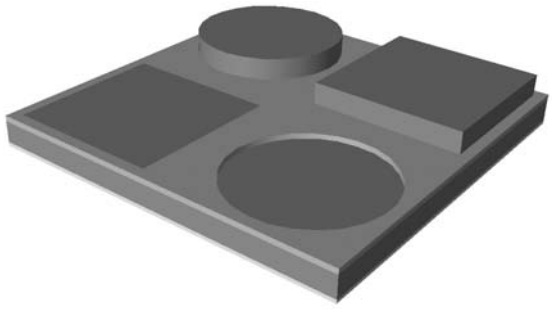
This exposed surface is etched on by comparing the exposed chunk’s level thickness with the etch depth. If the etch depth is equal to the level thickness the material of the chunk is changed to the ”Void” material. If the etch depth is less than the level thickness the level is split. The material of the new SVChunks that are created from the split and border the ”Void” chunks are changed to ”Void” material. If the etch depth is greater than the level thickness the material of the chunk is changed to ”Void” material. The remaining etch depth, calculated by subtracting the level thickness from the etch depth, is used to continue etching into the chunks above or below the new exposed surface.

This method of simulating a wet etch can produce some inaccuracies. Specifically, there is a possiblity that a chunk of affected material that is not reachable may still be etched. This happens during the side etch step of the process. A chunk of affected material that is unreachable will still be etched if it is within the etch depth of a reachable chunk of affected material. This will happen even if there is a barrier of non affected material between the two chunks. The reason for this is that the surface created from offsetting the virtual surface of the ajacent chunk cluster is imprinted into the complete surface list of the SVMEMSGeometry. Thus no distinction is made between the chunks that are reachable and those that are not. To correct this problem a reachability method would need to be designed that took into account those chunks that are being etched as well as the current ”Void” chunks.

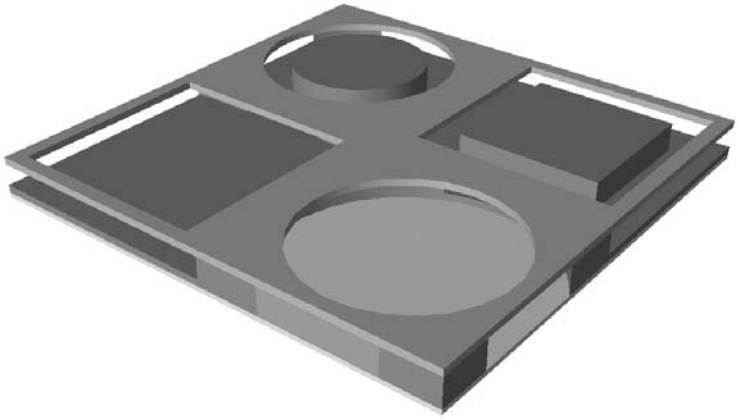
Figure 2.5(a) shows an existing $2\frac{1}{2}$D geometry consisting of three different materials. Figure 2.5(b) shows the results of a wet etch affecting one material. Each exposed surface is etched. The surfaces that are on border of the geometry’s bounding box are not considered exposed.

2.3.6 Release Etch

The purpose of this class is to simulate a release etch. This etch is performed by finding all SVChunks of affected material and that are reachable from the top of the SVMEMSGeometry. The material of these chunks is changed from their present material to ”Void” material.

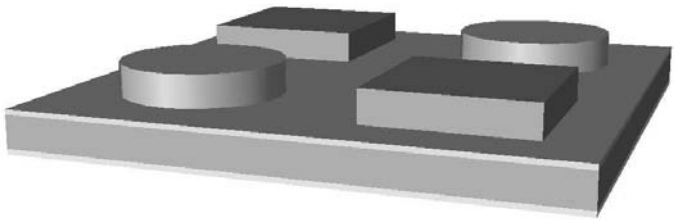


(a) Before Wet Etch.

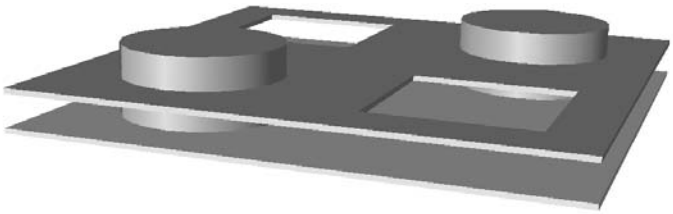


(b) After Wet Etch.

Figure 2.5. Wet Etch of one material.



(a) Before Release Etch.



(b) After Release Etch.

Figure 2.6. Release Etch of one material.

Figure 2.6(a) shows an existing $2\frac{1}{2}$D geometry consisting of three different materials. Figure 2.6(b) shows the results of a release etch affecting one material. All affected material that is reachable from the top of the geometry is removed. Material is not considered reachable if is only reachable from the surfaces that border the bounding box of the geometry.

Chapter 3

$2\frac{1}{2}$D Geometry Classes

This chapter describes the interfaces for the geometric data classes. Each section contains a general description of a class and a list of its interface methods. A description of each interface method is then provided.

3.1 SVChunk Class Reference

SVChunk is a typedef for std::pair<unsigned int,SVMaterial*>.

`#include <SVLevel.hpp>`

3.1.1 Detailed Description

SVChunk is a typedef for std::pair<unsigned int,SVMaterial*>.

The unsigned long is a pointer to an instance of DDSurface. Each SVChunk represents a 2 1/2d geometry and a material. The geometry is defined by an instance of DDSurface and the z value of the **SVLevel**(p. 44) where it resides. The material is defined by an instance of **SVMaterial**(p. 38).

3.2 SVMaterial Class Reference

This class is used to represent the various materials used to create a MEMS geometry. Each material instance has a unique name which is made up of the root material name and the process step number at which the material was deposited.

`#include <SVMaterial.hpp>`

Public Member Functions

- **SVMaterial** (std::string &r_material_name, unsigned long step_number)
- ~**SVMaterial** ()
- DD_RESULT **get_material_name** (std::string &r_material_name)
- std::string **get_name** ()
- DD_RESULT **get_ID** (unsigned long &r_id)
- unsigned long **get_ID** ()
- std::string * **root_material** ()

Static Public Member Functions

- unsigned long **get_ID_generator** ()
- DD_RESULT **get_material** (std::string &r_name, **SVMaterial** *&rp_material)
- DD_RESULT **get_material** (unsigned long &r_id, **SVMaterial** *&rp_material)
- int **get_instance_map_size** ()
- void **delete_all_instances** ()
- void **get_deposition_materials** (std::list< **SVMaterial** * > &r_material_list)
- void **add_deposition_material** (**SVMaterial** *p_material)
- void **get_root_materials** (std::list< std::string * > &r_material_list)

Public Attributes

- **SVColor** * **mColor**
- std::string **msProcessStepName**
- unsigned long **mStepID**

3.2.1 Detailed Description

This class is used to represent the various materials used to create a MEMS geometry. Each material instance has a unique name which is made up of the root material name and the process step number at which the material was deposited.

Author:
Corey McBride

Date:
01/10/03

3.2.2 Constructor & Destructor Documentation

3.2.2.1 SVMaterial::SVMaterial (std::string & *r_material_name*, unsigned long *step_number*)

Constructor.

Parameters:

r_material_name The name of the material. This name needs to be unique from any other material created. If two materials have the same name then the second material will not be created correctly.

step_number The step number at which this material is first deposited. Each <material_name,step_number> pair created must be unique.

3.2.2.2 SVMaterial::~SVMaterial ()

Destructor Removes the SVMaterial from the list of all SVMaterials.

3.2.3 Member Function Documentation

3.2.3.1 void SVMaterial::add_deposition_material (SVMaterial * *p_material*) [static]

This function adds deposition material to the list of depositon materials.

Parameters:

p_material The material to add.

3.2.3.2 void SVMaterial::delete_all_instances () [static]

This function deletes all instance of the class. This function should only be used to clean up memory at the end of a program.

3.2.3.3 void SVMaterial::get_deposition_materials (std::list< SVMaterial * > & *r_material_list*) [static]

This function returns the deposition materials that have bee added.

Parameters:

r_material_list The list of materials returned.

3.2.3.4 unsigned long SVMaterial::get_ID ()

Returns the id of this instance.

3.2.3.5 DD_RESULT SVMaterial::get_ID (unsigned long & *r_id*)

Gets the ID of this instance.

Parameters:

r_id The variable used to return the id of this instance.

3.2.3.6 unsigned long SVMaterial::get_ID_generator () [static]

This method returns the ID of the last SVMaterial created. A new instance that is created and assigned an id manually should be assigned an id that is greater that the returned value.

3.2.3.7 int SVMaterial::get_instance_map_size () [static]

Returns the number of instances of the class. See the description of the constructor for more information on which instances are saved in the instance map.

3.2.3.8 DD_RESULT SVMaterial::get_material (unsigned long & *r_id*, SVMaterial *& *rp_material*) [static]

Get the pointer to the SVMaterial with the id *r_id*.

Parameters:

r_id The id of the SVMaterial to get.

rp_material The pointer where the SVMaterial is returned.

Return values:

DD_SUCCESS If the SVMaterial was found.

DD_ERROR_PRIMITIVE_NOT_FOUND If a SVMaterial was not found with the id *r_id*.

3.2.3.9 DD_RESULT SVMaterial::get_material (std::string & *r_name*, SVMaterial *& *rp_material*) [static]

Get the pointer to the SVMaterial with the name *r_name*.

Parameters:

r_name The name of the SVMaterial to get.

rp_material The pointer where the SVMaterial is returned.

Return values:

DD_SUCCESS If the SVMaterial was found.

DD_ERROR_PRIMITIVE_NOT_FOUND If a SVMaterial was not found with the name *r_name*.

3.2.3.10 DD_RESULT SVMaterial::get_material_name (std::string & *r_material_name*)

Gets the std::string name of this intance.

Parameters:

r_material_name The variable used to return the name of this instance.

3.2.3.11 std::string SVMaterial::get_name ()

Returns the name of this instance.

3.2.3.12 void SVMaterial::get_root_materials (std::list< std::string * > & *r_material_list*) [static]

This function returns a list of all of the root materials.

Parameters:

r_material_list The list in which the root materials will be returned.

3.2.3.13 std::string* SVMaterial::root_material ()

This function returns the root material for a material.

Return values:

The name of the root material

3.2.4 Member Data Documentation

3.2.4.1 SVColor* SVMaterial::mColor

The color of the deposition material.

3.2.4.2 std::string SVMaterial::msProcessStepName

The process step name for which the material is deposited.

3.2.4.3 unsigned long SVMaterial::mStepID

Step number at which the material was deposited.

3.3 SVColor Class Reference

SVColor holds the color number and rgb information.

`#include <SVColor.hpp>`

Public Attributes

- int **mColorNumber**
- double **mRed**
- double **mGreen**
- double **mBlue**

3.3.1 Detailed Description

SVColor holds the color number and rgb information.

3.3.2 Member Data Documentation

3.3.2.1 double SVColor::mBlue

The blue value of the RGB color.

3.3.2.2 int SVColor::mColorNumber

The color number as found in the process definition file.

3.3.2.3 double SVColor::mGreen

The green value of the RGB color.

3.3.2.4 double SVColor::mRed

The red value of the RGB color.

3.4 SVLevel Class Reference

This class is used to store all of the SVChunks that reside at a particular level in the **SVMemsGeometry**(p. 49). This class is derived from DDHashMap. Thus this class has all the functionality of the DDHashMap.

.

`#include <SVLevel.hpp>`

Public Member Functions

- **SVLevel** (unsigned long id=0xffffffff)
- ∼**SVLevel** ()
- DD_RESULT **register_mems_geometry** (**SVMemsGeometry** ∗p_mems_geometry, SVMemsGeometry::iterator geometry_iterator)
- DD_RESULT **unregister_mems_geometry** (**SVMemsGeometry** ∗p_mems_geometry)
- void **get_mems_geometry** (**SVMemsGeometry** ∗&rp_mems_geometry)
- SVMemsGeometry::iterator **get_mems_geometry_iterator** ()
- double **get_z_value** ()
- void **set_z_value** (double z_value)
- DD_RESULT **get_ID** (unsigned long &r_id)
- unsigned long **get_ID** ()

Static Public Member Functions

- void **delete_level** (**SVLevel** ∗&rp_level)
- unsigned long **get_ID_generator** ()
- DD_RESULT **get_level** (unsigned long &r_id, **SVLevel** ∗&rp_level)
- void **get_instance_map** (DDHashMap< unsigned long, **SVLevel** ∗ > &r_level_map)
- int **get_instance_map_size** ()
- void **delete_all_instances** ()

3.4.1 Detailed Description

This class is used to store all of the SVChunks that reside at a particular level in the **SVMemsGeometry**(p. 49). This class is derived from DDHashMap. Thus this class has all the functionality of the DDHashMap.

.

Author:
Corey McBride

Date:
05/15/03

The key into the map is an unsigned long. This key is intended to be a pointer to an instance of DDSurface. The instance of DDSurface should also be located in the surface list of the instance of **SVMemsGeometry**(p. 49) where the SVLevel resides. The data for the map is a pointer to an instance of **SVMaterial**(p. 38). Together the key and data from a std::pair. A **SVChunk**(p. 37) is a typedef for this particular std::pair.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 SVLevel::SVLevel (unsigned long *id* = `0xffffffff`)

Constructor.

Parameters:

id If *id* is 0xffffffff then the instance will be given a unique ID based upon the order of creation of the instance. The SVLevel will also be added to the list of all SVLevel. If *id* is 0 then the SVLevel will not be added to the static list of all SVLevels and will not be given a unique id. Instead the instance will have an id of 0. If *id* is between 0 and 0xffffffff then the instance will be assigned the ID *id*. However, the IDs must be assigned in assending order or the constructor will assert.

3.4.2.2 SVLevel::∼SVLevel ()

Destructor Removes the SVLevel from the list of all SVLevels.

3.4.3 Member Function Documentation

3.4.3.1 void SVLevel::delete_all_instances () `[static]`

This function deletes all instance of the class. This function should only be used to clean up memory at the end of a program. Higher order objects should be deleted first to make sure that this instance is not being used by another class.

3.4.3.2 void SVLevel::delete_level (SVLevel *& *rp_level*) [static]

Deletes the SVLevel.

Parameters:

rp_level The SVLevel to be deleted.

3.4.3.3 unsigned long SVLevel::get_ID ()

Returns the id of the instance.

3.4.3.4 DD_RESULT SVLevel::get_ID (unsigned long & *r_id*)

Get the id of the instance.

Parameters:

r_id The variable where the id is returned.

3.4.3.5 unsigned long SVLevel::get_ID_generator () [static]

This method returns the ID of the last SVLevel created. A new instance that is created and assigned an id manually should be assigned an id that is greater that the returned value.

3.4.3.6 void SVLevel::get_instance_map (DDHashMap< unsigned long, SVLevel * > & *r_level_map*) [static]

Get a map containing pointers to all of the instances of the class indexed by instance id. See the description of the constructor for more information on which instances are saved in the instance map.

Parameters:

r_level_map The map where the pointers are returned.

3.4.3.7 int SVLevel::get_instance_map_size () [static]

Returns the number of instances of the class. See the description of the constructor for more information on which instances are saved in the instance map.

3.4.3.8 DD_RESULT SVLevel::get_level (unsigned long & *r_id*, SVLevel *& *rp_level*) [static]

Get the SVLevel with the id *r_id*.

Parameters:

r_id The id of the SVLevel to get.

rp_level The pointer where the SVLevel is returned.

Return values:

DD_SUCCESS If the SVLevel was found.

DD_ERROR_PRIMITIVE_NOT_FOUND If a SVLevel was not found with the id *r_id*.

3.4.3.9 void SVLevel::get_mems_geometry (SVMemsGeometry *& *rp_mems_geometry*)

Get the **SVMemsGeometry**(p. 49) that uses this instance.

Parameters:

rp_mems_geometry The **SVMemsGeometry**(p. 49) that uses this instance. If the value is NULL then the level is not being used in any **SVMemsGeometry**(p. 49).

3.4.3.10 SVMemsGeometry::iterator SVLevel::get_mems_geometry_iterator ()

Returns the interator into the **SVMemsGeometry**(p. 49) that uses this instance. This is only valid if the SVLevel belongs to a **SVMemsGeometry**(p. 49).

3.4.3.11 double SVLevel::get_z_value ()

Returns the Z value for this level.

3.4.3.12 DD_RESULT SVLevel::register_mems_geometry (SVMemsGeometry * *p_mems_geometry*, SVMemsGeometry::iterator *geometry_iterator*)

Registers the **SVMemsGeometry**(p. 49) that uses this instance. The method does not notify the **SVMemsGeometry**(p. 49) that it was registered with the SVLevel. Because of the 2d nature of the domain, each SVLevel may only be used once.

Parameters:

p_mems_geometry The pointer to the **SVMemsGeometry**(p. 49) that uses the SVLevel.

geometry_iterator The SVMemsGeometry::iterator for the location of the SVLevel in the **SVMemsGeometry**(p. 49).

Return values:

DD_SUCCESS If the p_mems_geometry was registered.

DD_FAILURE If the SVLevel is already being used by another **SVMemsGeometry**(p. 49).

3.4.3.13 void SVLevel::set_z_value (double *z_value*)

Sets the Z value for this SVLevel.

Parameters:

z_value The Z value for this SVLevel.

3.4.3.14 DD_RESULT SVLevel::unregister_mems_geometry (SVMemsGeometry * *p_mems_geometry*)

Unregisters the provided **SVMemsGeometry**(p. 49) that no longer uses this instance. SVLevel does not notify the **SVMemsGeometry**(p. 49) that it was unregistered.

Parameters:

p_mems_geometry The **SVMemsGeometry**(p. 49) to be unregistered.

Return values:

DD_SUCCESS If the p_mems_geometry was unregistered.

DD_ERROR_PRIMITIVE_NOT_FOUND If the **SVMemsGeometry**(p. 49) is not using this instance of SVLevel.

3.5 SVMemsGeometry Class Reference

This class is used to store all of the SVLevels that make up a MEMS part.

`#include <SVMemsGeometry.hpp>`

Public Member Functions

- **SVMemsGeometry** (DDBoundingBox &r_bounding_box, unsigned long id=0xffffffff)
- ~**SVMemsGeometry** ()
- void **push_front** (**SVLevel** *&rp_level)
- void **push_back** (**SVLevel** *&rp_level)
- void **pop_front** ()
- void **pop_back** ()
- std::list< **SVLevel** * >::iterator **insert** (std::list< **SVLevel** * >::iterator pos, **SVLevel** *&rp_level)
- std::list< **SVLevel** * >::iterator **erase** (std::list< **SVLevel** * >::iterator pos)
- void **remove** (**SVLevel** *p_level)
- void **clear** ()
- DD_RESULT **get_ID** (unsigned long &r_id)
- DDBoundingBox **get_bounding_box** ()
- void **set_bounding_box** (DDBoundingBox &r_bounding_box)
- void **set_tolerance** (double tolerance)
- double **get_tolerance** ()
- double **get_height** ()

Static Public Member Functions

- DD_RESULT **get_mems_geometry** (unsigned long &r_id, **SVMemsGeometry** *&rp_mems_geometry)
- DD_RESULT **get_instance_map** (DDHashMap< unsigned long, **SVMemsGeometry** * > &r_mems_geometry_map)
- int **get_instance_map_size** ()
- void **delete_all_instances** ()
- unsigned long **get_ID_generator** ()

Public Attributes

- std::list< DDSurface * > **mSurfaceList**

Protected Attributes

- unsigned long **mID**

Static Protected Attributes

- unsigned long **msIDGenerator**

3.5.1 Detailed Description

This class is used to store all of the SVLevels that make up a MEMS part.

Author:
Corey McBride

Date:
05/15/03

This class is derived from std::list so it has all of the functionality of a std::list. The order of the SVLevels in the SVMemsGeometry is important. The first **SVLevel**(p. 44) is the top of the geometry while the last **SVLevel**(p. 44) is the bottom of the geometry.

A sub set of std::list functions have been overloaded to automatically maintain the link from **SVLevel**(p. 44) to SVMemsGeometry. These function deal with adding and removing **SVLevel**(p. 44) from the SVMemsGeometry. If these overloaded functions are used the link will be maintained automatically. Otherwise it is the developers responsiblity to maintain the link. All of the other functions and iterators for std:list work as designed. This was done so that it would not be necessary to create a complete set of interface functions to the list class.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 SVMemsGeometry::SVMemsGeometry (DDBoundingBox & *r_bounding_box*, unsigned long *id* = `0xffffffff`)

Constructor.

Parameters:

id If *id* is 0xffffffff then the instance will be given a unique ID based upon the order of creation of the instance. The SVMemsGeometry will also be added to the list of all SVMems-Geometry. If *id* is 0 then the SVMemsGeometry will not be added to the static list of

all SVMemsGeometrys and will not be given a unique id. Instead the instance will have an id of 0. If *id* is between 0 and 0xffffffff then the instance will be assigned the ID *id*. However, the IDs must be assigned in assending order or the constructor will assert.

r_bounding_box The bounding box of the future geometry.

3.5.2.2 SVMemsGeometry::~SVMemsGeometry ()

Destructor. Removes the SVMemsGeometry from the list of all SVMemsGeometrys.

3.5.3 Member Function Documentation

3.5.3.1 void SVMemsGeometry::clear ()

Remove all of the SVLevels from the SVMemsGeometry. This method also unregisters the SVMemsGeometry with the SVLevels.

3.5.3.2 void SVMemsGeometry::delete_all_instances () [static]

This function deletes all instance of the class. This function should only be used to clean up memory at the end of a program.

3.5.3.3 std::list<SVLevel*>::iterator SVMemsGeometry::erase (std::list< SVLevel * >::iterator *pos*)

Erases the **SVLevel**(p. 44) represented by the iterator *pos* from the SVMemsGeometry. This method also unregisters the SVMemsGeometry with the **SVLevel**(p. 44).

Parameters:

pos The iterator for the **SVLevel**(p. 44) to remove.

Returns:

The iterator to the **SVLevel**(p. 44) immediately following the one removed.

3.5.3.4 DDBoundingBox SVMemsGeometry::get_bounding_box ()

Returns the DDBoundingBox for the geometry. This method doesn't calculate the bounding box of the geometry.

Return values:

The DDBoundingBox of the geometry.

3.5.3.5 double SVMemsGeometry::get_height ()

Gets the height of the geometry. The height is calculated by adding the height of each **SVLevel**(p. 44) in the geometry.

Returns:

The height of the geometry.

3.5.3.6 DD_RESULT SVMemsGeometry::get_ID (unsigned long & *r_id*)

Get the id of the instance.

Parameters:

r_id The variable where the id is returned.

3.5.3.7 unsigned long SVMemsGeometry::get_ID_generator () [static]

This method returns the ID of the last SVMemsGeometry created. A new instance that is created and assigned an id manually should be assigned an id that is greater that the returned value.

3.5.3.8 DD_RESULT SVMemsGeometry::get_instance_map (DDHashMap< unsigned long, SVMemsGeometry * > & *r_mems_geometry_map*) [static]

Get a map containing pointers to all of the instances of the class indexed by instance id. See the description of the constructor for more information on which instances are saved in the instance map.

Parameters:

r_mems_geometry_map The map where the pointers are returned.

3.5.3.9 int SVMemsGeometry::get_instance_map_size () [static]

Returns the number of instances of the class. See the description of the constructor for more information on which instances are saved in the instance map.

3.5.3.10 DD_RESULT SVMemsGeometry::get_mems_geometry (unsigned long & *r_id*, SVMemsGeometry *& *rp_mems_geometry*) [static]

Get the SVMemsGeometry with the id *r_id*.

Parameters:

r_id The id of the SVMemsGeometry to get.

rp_mems_geometry The variable where the SVMemsGeometry is to be returned.

Return values:

DD_SUCCESS If the SVMemsGeometry was found.

DD_ERROR_PRIMITIVE_NOT_FOUND If a SVMemsGeometry was not found with the id *r_id*.

3.5.3.11 double SVMemsGeometry::get_tolerance ()

Returns the tolerance used for the geometry.

3.5.3.12 std::list<SVLevel*>::iterator SVMemsGeometry::insert (std::list< SVLevel * >::iterator *pos*, SVLevel *& *rp_level*)

Inserts the **SVLevel**(p. 44) before the position indicated by *pos*. This method also registers the SVMemsGeometry with the **SVLevel**(p. 44). The **SVLevel**(p. 44) is not inserted if it is being used by another instance of SVMemsGeometry.

Parameters:

pos The SVMemsGeometry iterator for the position to insert the **SVLevel**(p. 44) before.

rp_level The **SVLevel**(p. 44) to insert.

Returns:

The iterator to the position of the **SVLevel**(p. 44) inserted.

3.5.3.13 void SVMemsGeometry::pop_back ()

Remove the **SVLevel**(p. 44) from the back of the SVMemsGeometry. This method also unregisters the SVMemsGeometry with the **SVLevel**(p. 44).

3.5.3.14 void SVMemsGeometry::pop_front ()

Remove the **SVLevel**(p. 44) from the front of the SVMemsGeometry. This method also unregisters the SVMemsGeometry with the **SVLevel**(p. 44).

3.5.3.15 void SVMemsGeometry::push_back (SVLevel ∗& *rp_level*)

Push a **SVLevel**(p. 44) to the back of the SVMemsGeometry. This means that the **SVLevel**(p. 44) pushed will be the bottom of the geometry. This method also registers the SVMemsGeometry with the **SVLevel**(p. 44). This method doesn't push the **SVLevel**(p. 44) if the **SVLevel**(p. 44) is already being used by another SVMemsGeometry.

Parameters:

rp_level The **SVLevel**(p. 44) to add to the SVMemsGeometry.

3.5.3.16 void SVMemsGeometry::push_front (SVLevel ∗& *rp_level*)

Push a **SVLevel**(p. 44) to the front of the SVMemsGeometry. This means that the **SVLevel**(p. 44) pushed will be the top of the geometry. This method also registers the SVMemsGeometry with the **SVLevel**(p. 44). This method doesn't push the **SVLevel**(p. 44) if the **SVLevel**(p. 44) is already being used by another SVMemsGeometry.

Parameters:

rp_level The **SVLevel**(p. 44) to add to the SVMemsGeometry.

3.5.3.17 void SVMemsGeometry::remove (SVLevel ∗ *p_level*)

Removes the **SVLevel**(p. 44) from the SVMemsGeometry. This method also unregisters the SVMemsGeometry with the **SVLevel**(p. 44).

Parameters:

p_level The **SVLevel**(p. 44) to remove.

3.5.3.18 void SVMemsGeometry::set_bounding_box (DDBoundingBox & *r_bounding_box*)

Sets the DDBoundingBox for the geometry. This method doesn't calculate the bounding box of the geometry.

Parameters:

r_bounding_box The DDBoundingBox for the geometry.

3.5.3.19 void SVMemsGeometry::set_tolerance (double *tolerance*)

Sets the tolerance for the geometry.

Parameters:

tolerance The tolerance used for the geometry.

3.5.4 Member Data Documentation

3.5.4.1 unsigned long SVMemsGeometry::mID [protected]

A unique identifier for each mems geometry.

3.5.4.2 unsigned long SVMemsGeometry::msIDGenerator [static, protected]

A static variable that is indexed with the creation of each instance of a.

3.5.4.3 std::list<DDSurface*> SVMemsGeometry::mSurfaceList

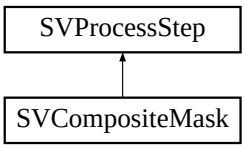
A list of DDSurfaces that make up the 2d geometry of the 2 1/2d SVMemsGeometry. The DDSurfaces need to be interconnected and tile the bounding box of the geometry. This member variable is public on purpose.

3.6 SVCompositeMask Class Reference

This class is used to store the DDSurfaces that define the geometry of a 2d mask set.

`#include <SVCompositeMask.hpp>`

Inheritance diagram for SVCompositeMask::



Public Member Functions

- **SVCompositeMask** (std::string &r_name)
- **SVCompositeMask** (**SVProcessStepData** &r_parse_step_data)
- **SVCompositeMask** (**SVCompositeMask** &r_composite_mask)
- DD_RESULT **set_field** (**SVFieldType** field)
- **SVFieldType get_field** ()
- std::string **get_name** ()
- ∼**SVCompositeMask** ()
- DD_RESULT **print_information** ()
- **SVProcessStep** ∗ **get_parent_process_step** ()
- DD_RESULT **set_parent_process_step** (**SVProcessStep** ∗p_parent_process_step)

3.6.1 Detailed Description

This class is used to store the DDSurfaces that define the geometry of a 2d mask set.

Author:
Corey McBride

Date:
05/15/03

The DDSurfaces stored in this class are the result of unioning all of the DDSurfaces together from one layer of the 2d mask set. The DDSurfaces are also the result of XORing two layers of the 2d mask set together.

The class is derived from std::list<DDSurface∗>. Thus this class has all the functionality of a std::list. The order of the DDSurfaces in the list in not important.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 SVCompositeMask::SVCompositeMask (std::string & *r_name*)

Constructor.

Parameters:

r_name The name of the mask.

3.6.2.2 SVCompositeMask::SVCompositeMask (SVProcessStepData & *r_parse_step_data*)

Constructor.

Parameters:

r_parse_step_data The instance of **SVProcessStepData**(p. 85) that contains the information necessary to initialize the instance.

3.6.2.3 SVCompositeMask::SVCompositeMask (SVCompositeMask & *r_composite_mask*)

Copy Constructor.

Parameters:

r_composite_mask An instance of SVCompositeMask to copy.

3.6.2.4 SVCompositeMask::~SVCompositeMask ()

Destructor.

Deletes all instances of DDSurface that are contained in the mask.

3.6.3 Member Function Documentation

3.6.3.1 SVFieldType SVCompositeMask::get_field ()

Returns the field of the mask.

3.6.3.2 std::string SVCompositeMask::get_name ()

Returns the name of the mask.

Reimplemented from **SVProcessStep** (p. 63).

3.6.3.3 SVProcessStep* SVCompositeMask::get_parent_process_step ()

Returns the **SVProcessStep**(p. 61) that uses this instance of SVCompositeMask.

3.6.3.4 DD_RESULT SVCompositeMask::print_information () [virtual]

Prints the information for this class to std::cout. This is an overloaded function from **SVProcessStep**(p. 61).

Reimplemented from **SVProcessStep** (p. 63).

3.6.3.5 DD_RESULT SVCompositeMask::set_field (SVFieldType *field*)

Sets the field of the mask.

Parameters:

field The SVFieldType object that contains the field for this mask.

3.6.3.6 DD_RESULT SVCompositeMask::set_parent_process_step (SVProcessStep * *p_parent_process_step*)

Sets the SVProcess that uses this instance of SVCompositeMask.

3.7 SVMGTChunkCluster Class Reference

This class represents a group of DDSurfaces that are interconnected.

`#include <SVModifyGeometryTool.hpp>`

3.7.1 Detailed Description

This class represents a group of DDSurfaces that are interconnected.

Author:
Corey McBride

Date:
06/02/03

Interconnected DDSurfaces share instances of DDEdge. This class is derived from a std::list. Thus this class has all of the functionality of a std::list.

Chapter 4

SUMMiT V Process Step Classes

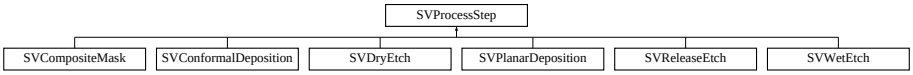
This chapter describes the interfaces for the process step classes. Each section contains a general description of a class and a list of its interface methods. A description of each interface method is then provided.

4.1 SVProcessStep Class Reference

SVProcessStep is the base class from which all process steps are derived.

`#include <SVProcessStep.hpp>`

Inheritance diagram for SVProcessStep::



Public Member Functions

- **SVProcessStep** (**SVProcessStepData** &r_parse_step_data)
- **SVProcessStep** ()
- ~**SVProcessStep** ()
- virtual DD_RESULT **execute** (**SVMemsGeometry** &r_mems_geometry)
- void **set_name** (std::string &r_name)
- void **set_process_type** (**SVProcessType** &r_type)
- unsigned long **get_order** ()
- std::string **get_name** ()
- **SVProcessType get_process_type** ()
- virtual DD_RESULT **print_information** ()

4.1.1 Detailed Description

SVProcessStep is the base class from which all process steps are derived.

Author:
Corey McBride

Date:
01/10/03

4.1.2 Constructor & Destructor Documentation

4.1.2.1 SVProcessStep::SVProcessStep (SVProcessStepData & *r_parse_step_data*)

Constructor.

Parameters:
r_parse_step_data The **SVProcessStepData**(p. 85) that contains the information necessary to initialize the SVProcessStep.

4.1.2.2 SVProcessStep::SVProcessStep ()

Constructor.

4.1.2.3 SVProcessStep::~SVProcessStep ()

Destructor.

4.1.3 Member Function Documentation

4.1.3.1 virtual DD_RESULT SVProcessStep::execute (SVMemsGeometry & *r_mems_geometry*) [inline, virtual]

The main routine to perform the process step on the **SVMemsGeometry**(p. 49). Each class derived from this base class defines this routine.

Parameters:
r_mems_geometry The **SVMemsGeometry**(p. 49) to apply the process step to.

Reimplemented in **SVConformalDeposition** (p. 72), **SVDryEtch** (p. 77), **SVPlanarDeposition** (p. 66), **SVPlanarEtch** (p. 69), **SVReleaseEtch** (p. 83), and **SVWetEtch** (p. 80).

4.1.3.2 std::string SVProcessStep::get_name ()

Returns the name of the SVProcessStep.

Returns:
The name of the SVProcessStep.

Reimplemented in **SVCompositeMask** (p. 58).

4.1.3.3 unsigned long SVProcessStep::get_order ()

Returns the order of the SVProcessStep.

Returns:
The order of the SVProcessStep.

4.1.3.4 SVProcessType SVProcessStep::get_process_type ()

Returns the type of the SVProcessStep.

Returns:
The type of the SVProcessStep.

4.1.3.5 virtual DD_RESULT SVProcessStep::print_information () [virtual]

Prints out the information for the SVProcessStep to std::cout.

Reimplemented in **SVCompositeMask** (p. 58), **SVConformalDeposition** (p. 73), **SVDryEtch** (p. 77), **SVPlanarDeposition** (p. 66), **SVPlanarEtch** (p. 69), and **SVWetEtch** (p. 81).

4.1.3.6 void SVProcessStep::set_name (std::string & *r_name*)

Sets the name of the SVProcessStep.

Parameters:
r_name The name of the SVProcessStep.

4.1.3.7 void SVProcessStep::set_process_type (SVProcessType & *r_type*)

Sets the SVProcessStep's type.

Parameters:

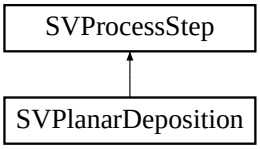
r_type The SVProcessType for the SVProcessStep.

4.2 SVPlanarDeposition Class Reference

This class defines the planar deposition process.

`#include <SVPlanarDeposition.hpp>`

Inheritance diagram for SVPlanarDeposition::



Public Member Functions

- **SVPlanarDeposition** ()
- **SVPlanarDeposition** (**SVProcessStepData** &r_parse_step_data)
- void **set_deposition_material** (**SVMaterial** *p_material)
- void **set_void_material** (**SVMaterial** *p_material)
- void **set_deposition_thickness** (double deposition_thickness)
- DD_RESULT **print_information** ()
- DD_RESULT **execute** (**SVMemsGeometry** &r_mems_geometry)

4.2.1 Detailed Description

This class defines the planar deposition process.

Author:
Corey McBride

Date:
05/29/03

The planar deposition process modifies the **SVMemsGeometry**(p. 49) by depositing material to fill in all reachable void chunks up to the deposition thickness. The thickness of the deposition is an absolute thickness from the bottom of the **SVMemsGeometry**(p. 49). Thus any structure that extends above this thickness is removed. Or in other words, the overall height of the **SVMemsGeometry**(p. 49) will be the specified deposition thickness of the planar deposition process.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 SVPlanarDeposition::SVPlanarDeposition ()

Constructor The deposition material, deposition thickness and void material must be specified for deposition to work properly.

4.2.2.2 SVPlanarDeposition::SVPlanarDeposition (SVProcessStepData & *r_parse_step_data*)

Constructor.

Parameters:

r_parse_step_data The instance of **SVProcessStepData**(p. 85) that contains the information to initialize this instance.

4.2.3 Member Function Documentation

4.2.3.1 DD_RESULT SVPlanarDeposition::execute (SVMemsGeometry & *r_mems_geometry*) [virtual]

Performs a planar deposition on the provided **SVMemsGeometry**(p. 49).

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to perform the planar deposition on.

Return values:

DD_X_CRITICAL_ERROR If the routine had an error that it wasn't able to recover from.

DD_X_NULL_POINTER If the void **SVMaterial**(p. 38) or deposition **SVMaterial**(p. 38) were not set.

DD_SUCCESS If the planar deposition was successfull.

Reimplemented from **SVProcessStep** (p. 62).

4.2.3.2 DD_RESULT SVPlanarDeposition::print_information () [virtual]

Prints the information about this process step. This is an overloaded function from **SVProcessStep**(p. 61).

Reimplemented from **SVProcessStep** (p. 63).

4.2.3.3 void SVPlanarDeposition::set_deposition_material (SVMaterial * *p_material*)

Sets the deposition material.

Parameters:

p_material The **SVMaterial**(p. 38) used to fill in the reachable void chunks

4.2.3.4 void SVPlanarDeposition::set_deposition_thickness (double *deposition_thickness*)

Sets the deposition thickness.

Parameters:

deposition_thickness The thickness of the deposition.

4.2.3.5 void SVPlanarDeposition::set_void_material (SVMaterial * *p_material*)

Sets the instance of **SVMaterial**(p. 38) that is the void material.

Parameters:

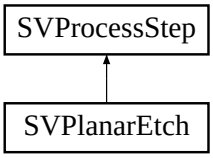
p_material The **SVMaterial**(p. 38) that is considered as void material during the deposition. The chunks of this **SVMaterial**(p. 38) are changed to the deposition **SVMaterial**(p. 38) by the process.

4.3 SVPlanarEtch Class Reference

This class defines the planar etch process.

`#include <SVPlanarEtch.hpp>`

Inheritance diagram for SVPlanarEtch::



Public Member Functions

- **SVPlanarEtch** ()
- **SVPlanarEtch** (**SVProcessStepData** &r_parse_step_data)
- void **set_void_material** (**SVMaterial** *p_material)
- void **set_etch_thickness** (double etch_thickness)
- DD_RESULT **print_information** ()
- DD_RESULT **execute** (**SVMemsGeometry** &r_mems_geometry)

4.3.1 Detailed Description

This class defines the planar etch process.

Author:
Corey McBride

Date:
05/29/03

The planar etch process modifies the **SVMemsGeometry**(p. 49) by removing all material above the etch thickness. The thickness of the etch is an absolute thickness from the bottom of the **SVMemsGeometry**(p. 49). Thus any structure that extends above this thickness is removed. Or in other words, the overall height of the **SVMemsGeometry**(p. 49) will be the specified etch thickness of the planar deposition process.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 SVPlanarEtch::SVPlanarEtch ()

Constructor The deposition material, deposition thickness and void material must be specified for deposition to work properly.

4.3.2.2 SVPlanarEtch::SVPlanarEtch (SVProcessStepData & *r_parse_step_data*)

Constructor.

Parameters:

r_parse_step_data The instance of **SVProcessStepData**(p. 85) that contains the information to initialize this instance.

4.3.3 Member Function Documentation

4.3.3.1 DD_RESULT SVPlanarEtch::execute (SVMemsGeometry & *r_mems_geometry*) [virtual]

Performs a planar etch on the provided **SVMemsGeometry**(p. 49).

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to perform the planar deposition on.

Return values:

DD_X_CRITICAL_ERROR If the routine had an error that it wasn't able to recover from.

DD_X_NULL_POINTER If the void **SVMaterial**(p. 38) was not set.

DD_SUCCESS If the planar etch was successfull.

Reimplemented from **SVProcessStep** (p. 62).

4.3.3.2 DD_RESULT SVPlanarEtch::print_information () [virtual]

Prints the information about this process step. This is an overloaded function from **SVProcessStep**(p. 61).

Reimplemented from **SVProcessStep** (p. 63).

4.3.3.3 void SVPlanarEtch::set_etch_thickness (double *etch_thickness*)

Sets the deposition thickness.

Parameters:

etch_thickness The thickness of the etch.

4.3.3.4 void SVPlanarEtch::set_void_material (SVMaterial * *p_material*)

Sets the instance of **SVMaterial**(p. 38) that is the void material.

Parameters:

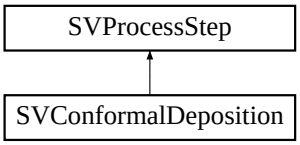
p_material The **SVMaterial**(p. 38) that is considered as void material during the deposition. The chunks of this **SVMaterial**(p. 38) are changed to the deposition **SVMaterial**(p. 38) by the process.

4.4 SVConformalDeposition Class Reference

This class defines a conformal deposition process. Conformal deposition modifies a **SVMemsGeometry**(p. 49) by offsetting the exposed surfaces with new material by the deposition thickness. Exposed surfaces are those surfaces that are reachable from the top most level by passing through the ”void” chunks. Because the domain is 2 1/2d the deposition in the X-Y plane is a rolling ball offset. The deposition in the Z direction is a straight offset.

`#include <SVConformalDeposition.hpp>`

Inheritance diagram for SVConformalDeposition::



Public Member Functions

- **SVConformalDeposition** ()
- **SVConformalDeposition** (**SVProcessStepData** &r_parse_step_data)
- void **set_deposition_material** (**SVMaterial** *p_material)
- void **set_void_material** (**SVMaterial** *p_material)
- void **set_deposition_thickness** (double deposition_thickness)
- void **set_egg_factor** (double top_egg_x, double top_egg_y, double bottom_egg_x, double bottom_egg_y)
- DD_RESULT **execute** (**SVMemsGeometry** &r_mems_geometry)
- DD_RESULT **print_information** ()

4.4.1 Detailed Description

This class defines a conformal deposition process. Conformal deposition modifies a **SVMemsGeometry**(p. 49) by offsetting the exposed surfaces with new material by the deposition thickness. Exposed surfaces are those surfaces that are reachable from the top most level by passing through the ”void” chunks. Because the domain is 2 1/2d the deposition in the X-Y plane is a rolling ball offset. The deposition in the Z direction is a straight offset.

Author:
Corey McBride

Date:
05/29/03

4.4.2 Constructor & Destructor Documentation

4.4.2.1 SVConformalDeposition::SVConformalDeposition ()

Constructor If this constructor is used the deposition material, void material and deposition thickness must be set for the process to work properly.

4.4.2.2 SVConformalDeposition::SVConformalDeposition (SVProcessStepData & *r_parse_step_data*)

Constructor.

Parameters:

r_parse_step_data The instance of **SVProcessStepData**(p. 85) that contains the information to initialize this instance.

4.4.3 Member Function Documentation

4.4.3.1 DD_RESULT SVConformalDeposition::execute (SVMemsGeometry & *r_mems_geometry*) [virtual]

Performs a conformal deposition on the provided **SVMemsGeometry**(p. 49).

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to perform the conformal deposition on.

Return values:

DD_X_CRITICAL_ERROR If the routine had an error that it wasn't able to recover from.

DD_X_NULL_POINTER If the void **SVMaterial**(p. 38) or deposition **SVMaterial**(p. 38) were not set.

DD_SUCCESS If the conformal deposition was successful.

Reimplemented from **SVProcessStep** (p. 62).

4.4.3.2 DD_RESULT SVConformalDeposition::print_information () [virtual]

Prints the information about this process step. This is an overloaded function from **SVProcessStep**(p. 61).

Reimplemented from **SVProcessStep** (p. 63).

4.4.3.3 void SVConformalDeposition::set_deposition_material (SVMaterial * *p_material*)

Sets the instance of **SVMaterial**(p. 38) that is the deposition material.

Parameters:

p_material The **SVMaterial**(p. 38) that is the deposition material.

4.4.3.4 void SVConformalDeposition::set_deposition_thickness (double *deposition_thickness*)

Sets the deposition thickness.

Parameters:

deposition_thickness The distance that the exposed surfaces are offset.

4.4.3.5 void SVConformalDeposition::set_egg_factor (double *top_egg_x*, double *top_egg_y*, double *bottom_egg_x*, double *bottom_egg_y*)

Sets the egg factors.

Parameters:

top_egg_x The top egg x deposition factor.

top_egg_y The top egg y deposition factor.

bottom_egg_x The bottom egg x deposition factor.

bottom_egg_y The bottom egg y deposition factor.

4.4.3.6 void SVConformalDeposition::set_void_material (SVMaterial * *p_material*)

Sets the instance of **SVMaterial**(p. 38) that is the void material. The void material is used to determine which surfaces are exposed.

Parameters:

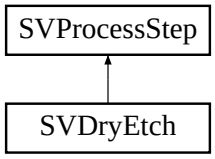
p_material The **SVMaterial**(p. 38) that is the void material.

4.5 SVDryEtch Class Reference

This class defines a dry etch process. The dry etch process modifies the **SVMemsGeometry**(p. 49) by etching the SVChunks of the affected **SVMaterial**(p. 38) based upon the geometry defined in the supplied **SVCompositeMask**(p. 56) object. The SVChunks are etched the etch depth straight down from the top **SVLevel**(p. 44). Thus, SVChunks of unaffected **SVMaterial**(p. 38) protect SVChunks of affected **SVMaterial**(p. 38) that lie below.

`#include <SVDryEtch.hpp>`

Inheritance diagram for SVDryEtch::



Public Member Functions

- **SVDryEtch** (**SVCompositeMask** *p_composite_mask)
- **SVDryEtch** (**SVProcessStepData** &r_parse_step_data, std::map< std::string, **SVCompositeMask** * > &r_composite_masks)
- ~**SVDryEtch** ()
- DD_RESULT **execute** (**SVMemsGeometry** &r_mems_geometry)
- DD_RESULT **set_etch_depth** (double etch_depth)
- DD_RESULT **set_over_etch** (double over_etch)
- DD_RESULT **set_etch_type** (**SVEtchType** etch_type)
- DD_RESULT **set_affected_materials** (std::list< **SVMaterial** * > &r_affected_materials)
- DD_RESULT **set_void_material** (**SVMaterial** *p_void_material)
- bool **is_affected_material** (**SVMaterial** *p_material)
- DD_RESULT **print_information** ()

4.5.1 Detailed Description

This class defines a dry etch process. The dry etch process modifies the **SVMemsGeometry**(p. 49) by etching the SVChunks of the affected **SVMaterial**(p. 38) based upon the geometry defined in the supplied **SVCompositeMask**(p. 56) object. The SVChunks are etched the etch depth straight down from the top **SVLevel**(p. 44). Thus, SVChunks of unaffected **SVMaterial**(p. 38) protect SVChunks of affected **SVMaterial**(p. 38) that lie below.

Author:
Corey McBride

Date:
05/29/03

4.5.2 Constructor & Destructor Documentation

4.5.2.1 SVDryEtch::SVDryEtch (SVCompositeMask * *p_composite_mask*)

Constructor This constructor requires the **SVCompositeMask**(p. 56) used in the etching. A **SVCompositeMask**(p. 56) must be provided even if it is an empty one. The etch depth, void material and other parameters must be set for the etching to work properly.

Parameters:
p_composite_mask The **SVCompositeMask**(p. 56) used in the etching.

4.5.2.2 SVDryEtch::SVDryEtch (SVProcessStepData & *r_parse_step_data*, std::map< std::string, SVCompositeMask * > & *r_composite_masks*)

Constructor.

This constructor requires the **SVCompositeMask**(p. 56) used in the etching. A **SVCompositeMask**(p. 56) must be provided even if it is an empty one. The etch depth, void material and other parameters are set by the instance of **SVProcessStepData**(p. 85).

Parameters:
r_parse_step_data The **SVProcessStepData**(p. 85) used to initialize the class.

r_composite_masks A map of **SVCompositeMask**(p. 56) objects that contains the **SVCompositeMask**(p. 56) used in the etching. Each instance of **SVCompositeMask**(p. 56) is indexed by mask name.

4.5.2.3 SVDryEtch::~SVDryEtch ()

Destructor

Deletes the **SVCompositeMask**(p. 56) used by the instance.

4.5.3 Member Function Documentation

4.5.3.1 DD_RESULT SVDryEtch::execute (SVMemsGeometry & *r_mems_geometry*) [virtual]

This routine performs the dry etch on the **SVMemsGeometry**(p. 49) provided.

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) that the dry etch is performed on.

Return values:

DD_X_CRITICAL_ERROR If the routine had an error that it wasn't able to recover from.

DD_X_NULL_POINTER If the void **SVMaterial**(p. 38) was not set or the **SVCompositeMask**(p. 56) was not set.

DD_FAILURE If no SVMaterials were set.

DD_SUCCESS If the dry etch was successful.

Reimplemented from **SVProcessStep** (p. 62).

4.5.3.2 bool SVDryEtch::is_affected_material (SVMaterial * *p_material*)

Determines if a material is one of the materials affected by the etch. This is determined by checking to see if the **SVMaterial**(p. 38) is in the instances affected material list.

Parameters:

p_material The **SVMaterial**(p. 38) to be checked.

Return values:

true If the **SVMaterial**(p. 38) is affected by the etch.

false If the **SVMaterial**(p. 38) is not affected by the etch.

4.5.3.3 DD_RESULT SVDryEtch::print_information () [virtual]

Prints the information about this process step. This is an overloaded function from **SVProcessStep**(p. 61).

Reimplemented from **SVProcessStep** (p. 63).

4.5.3.4 DD_RESULT SVDryEtch::set_affected_materials (std::list< SVMaterial * > & *r_affected_materials*)

Sets the materials to be etched.

Parameters:

r_affected_materials The list of **SVMaterial**(p. 38) instances that are affected by the etch.

4.5.3.5 DD_RESULT SVDryEtch::set_etch_depth (double *etch_depth*)

Sets the etch depth.

Parameters:

etch_depth The depth of the etch.

4.5.3.6 DD_RESULT SVDryEtch::set_etch_type (SVEtchType *etch_type*)

Sets the etch type.

Parameters:

etch_type The SVEtchType object that contains the etch type for this process. The SVEtchType and SVFieldType are used to determine if the DDSurfaces in the provided SVCompositeMasks represent areas to etch or to protect from etching.

4.5.3.7 DD_RESULT SVDryEtch::set_over_etch (double *over_etch*)

Sets the over etch factor.

Parameters:

over_etch The over etch factor. The etch_depth is multiplied by this value to determine the overall etch depth.

4.5.3.8 DD_RESULT SVDryEtch::set_void_material (SVMaterial * *p_void_material*)

Sets the instance of **SVMaterial**(p. 38) that is the void material.

Parameters:

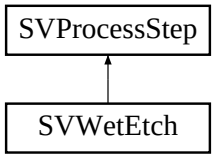
p_void_material The **SVMaterial**(p. 38) that etched material is changed too.

4.6 SVWetEtch Class Reference

This class defines a wet etch process.

`#include <SVWetEtch.hpp>`

Inheritance diagram for SVWetEtch::



Public Member Functions

- **SVWetEtch** ()
- **SVWetEtch** (**SVProcessStepData** &r_parse_step_data)
- DD_RESULT **execute** (**SVMemsGeometry** &r_mems_geometry)
- DD_RESULT **set_etch_depth** (double etch_depth)
- DD_RESULT **set_affected_materials** (std::list< **SVMaterial** * > &r_affected_materials)
- DD_RESULT **set_void_material** (**SVMaterial** *p_void_material)
- bool **is_affected_material** (**SVMaterial** *p_material)
- DD_RESULT **print_information** ()

4.6.1 Detailed Description

This class defines a wet etch process.

Author:
Corey McBride

Date:
05/29/03

The wet etch process modifies the **SVMemsGeometry**(p. 49) by etching all reachable SVChunks of the affected SVMaterials the specified depth. The etch starts at the face of each exposed **SVChunk**(p. 37). Thus, a cantilevered **SVChunk**(p. 37) of affected material will be etched the specified depth on each exposed face. The etch is performed by creating a volume that represents the reachable void SVChunks expanded the etch depth. Any SVChunks of affected material

within this volume are changed to the void **SVMaterial**(p. 38). This method can produce some inaccuracies in the resulting **SVMemsGeometry**(p. 49). Specifically, some SVChunks of affected material that are not reachable may be etched if they are within etch depth of void SVChunks that are reachable. In testing actual occurances of these inaccuracies appeared to be rare.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 SVWetEtch::SVWetEtch ()

Constructor The affected SVMaterials, void **SVMaterial**(p. 38) and etch depth need to be specified for this process to work correctly.

4.6.2.2 SVWetEtch::SVWetEtch (SVProcessStepData & *r_parse_step_data*)

Constructor.

Parameters:

r_parse_step_data The instance of **SVProcessStepData**(p. 85) that contains the information to initialize this instance.

4.6.3 Member Function Documentation

4.6.3.1 DD_RESULT SVWetEtch::execute (SVMemsGeometry & *r_mems_geometry*) [virtual]

Performs a wet etch on the provided **SVMemsGeometry**(p. 49).

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to perform the wet etch on.

Return values:

DD_X_CRITICAL_ERROR If the routine had an error that it wasn't able to recover from.

DD_X_NULL_POINTER If the void **SVMaterial**(p. 38) was not set.

DD_FAILURE If the affected SVMaterials are not specified.

DD_SUCCESS If the wet etch was successful.

Reimplemented from **SVProcessStep** (p. 62).

4.6.3.2 bool SVWetEtch::is_affected_material (SVMaterial * *p_material*)

Determines if a **SVMaterial**(p. 38) is one of the SVMaterials affected by the etch.

Parameters:

p_material The **SVMaterial**(p. 38) to be checked.

Return values:

true If the **SVMaterial**(p. 38) is affected by the etch.

false If the **SVMaterial**(p. 38) is not affected by the etch.

4.6.3.3 DD_RESULT SVWetEtch::print_information () [virtual]

Prints the information about this process step. This is an overloaded function from **SVProcessStep**(p. 61).

Reimplemented from **SVProcessStep** (p. 63).

4.6.3.4 DD_RESULT SVWetEtch::set_affected_materials (std::list< SVMaterial * > & *r_affected_materials*)

Sets the SVMaterials to be etched.

Parameters:

r_affected_materials The list of **SVMaterial**(p. 38) instances that are affected by the etch.

4.6.3.5 DD_RESULT SVWetEtch::set_etch_depth (double *etch_depth*)

Sets the etch depth.

Parameters:

etch_depth The depth to perform the etch.

4.6.3.6 DD_RESULT SVWetEtch::set_void_material (SVMaterial * *p_void_material*)

Sets the instance of **SVMaterial**(p. 38) that is the void material.

Parameters:

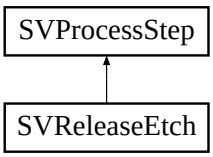
p_void_material The **SVMaterial**(p. 38) that the SVChunks of the affected **SVMaterial**(p. 38) are changed to.

4.7 SVReleaseEtch Class Reference

This class defines a release etch process.

`#include <SVReleaseEtch.hpp>`

Inheritance diagram for SVReleaseEtch::



Public Member Functions

- **SVReleaseEtch** ()
- **SVReleaseEtch** (**SVProcessStepData** &r_parse_step_data)
- DD_RESULT **set_affected_materials** (std::list< **SVMaterial** * > &r_affected_materials)
- bool **is_affected_material** (**SVMaterial** *p_material)
- DD_RESULT **set_void_material** (**SVMaterial** *p_void_material)
- DD_RESULT **execute** (**SVMemsGeometry** &r_mems_geometry)

4.7.1 Detailed Description

This class defines a release etch process.

Author:
Corey McBride

Date:
05/29/03

Release etch modifies the **SVMemsGeometry**(p. 49) by changing all reachable chunks of the specified **SVMaterial**(p. 38) to void **SVMaterial**(p. 38)

4.7.2 Constructor & Destructor Documentation

4.7.2.1 SVReleaseEtch::SVReleaseEtch ()

Constructor The affected SVMaterials and the void **SVMaterial**(p. 38) need to be specified for this process to work correctly.

4.7.2.2 SVReleaseEtch::SVReleaseEtch (SVProcessStepData & *r_parse_step_data*)

Constructor.

Parameters:

r_parse_step_data The instance of **SVProcessStepData**(p. 85) that contains the information to initialize this instance.

4.7.3 Member Function Documentation

4.7.3.1 DD_RESULT SVReleaseEtch::execute (SVMemsGeometry & *r_mems_geometry*) [virtual]

This routine performs a release etch on the **SVMemsGeometry**(p. 49) provided.

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) that the release etch is performed on.

Return values:

DD_X_CRITICAL_ERROR If the routine had an error that it wasn't able to recover from.

DD_X_NULL_POINTER If the void **SVMaterial**(p. 38) was not set.

DD_FAILURE If no affected SVMaterials were set.

DD_SUCCESS If the release etch was successful.

Reimplemented from **SVProcessStep** (p. 62).

4.7.3.2 bool SVReleaseEtch::is_affected_material (SVMaterial * *p_material*)

Determines if a **SVMaterial**(p. 38) is one of the SVMaterials affected by the etch.

Parameters:

p_material The **SVMaterial**(p. 38) to be checked.

Return values:

true If the **SVMaterial**(p. 38) is affected by the etch.

false If the **SVMaterial**(p. 38) is not affected by the etch.

4.7.3.3 DD_RESULT SVReleaseEtch::set_affected_materials (std::list< SVMaterial * > & *r_affected_materials*)

Sets the **SVMaterial**(p. 38) to be etched.

Parameters:

r_affected_materials The list of **SVMaterial**(p. 38) instances that are affected by the etch. The chunks of these SVMaterials are changed to the void **SVMaterial**(p. 38).

4.7.3.4 DD_RESULT SVReleaseEtch::set_void_material (SVMaterial * *p_void_material*)

Sets the instance of **SVMaterial**(p. 38) that is the void material.

Parameters:

p_void_material The **SVMaterial**(p. 38) that the SVChunks of the affected **SVMaterial**(p. 38) are changed to.

4.8 SVProcessStepData Class Reference

SVProcessStepData is used to store the information for a process step that is parsed from the process definition file.

`#include <SVProcessStep.hpp>`

Public Member Functions

- **SVProcessStepData** ()

Public Attributes

- unsigned int **mNumber**
- std::string **mAcadName**
- std::string **mName**
- **SVProcessType mType**
- double **mThickness**
- std::list< std::string > **mAffectedMaterials**
- std::string **mColor**
- std::string **mEdges**
- **SVEtchType mEtchType**
- **SVFieldType mFieldType**
- double **mOverEtch**
- double **mAngle**
- double **mTopEggX**
- double **mTopEggY**
- double **mBottomEggX**
- double **mBottomEggY**

4.8.1 Detailed Description

SVProcessStepData is used to store the information for a process step that is parsed from the process definition file.

Author:
Corey McBride

Date:
01/10/03

4.8.2 Constructor & Destructor Documentation

4.8.2.1 SVProcessStepData::SVProcessStepData () [inline]

Constructor.

4.8.3 Member Data Documentation

4.8.3.1 std::string SVProcessStepData::mAcadName

The AutoCAD layer name.

4.8.3.2 std::list<std::string> SVProcessStepData::mAffectedMaterials

The materials affected by the process step.

4.8.3.3 double SVProcessStepData::mAngle

The Angle of the etch. Not used for 2 1/2d geometry.

4.8.3.4 double SVProcessStepData::mBottomEggX

The bottom egg x factor.

4.8.3.5 double SVProcessStepData::mBottomEggY

The bottom egg y factor.

4.8.3.6 std::string SVProcessStepData::mColor

The color assigned for the process step.

4.8.3.7 std::string SVProcessStepData::mEdges

The mask name that is used by the process step.

4.8.3.8 SVEtchType SVProcessStepData::mEtchType

The type of etch.

4.8.3.9 SVFieldType SVProcessStepData::mFieldType

The field type of the etch.

4.8.3.10 std::string SVProcessStepData::mName

The process step name.

4.8.3.11 unsigned int SVProcessStepData::mNumber

The process step number.

4.8.3.12 double SVProcessStepData::mOverEtch

The over etch factor.

4.8.3.13 double SVProcessStepData::mThickness

The Thickness used my the process step.

4.8.3.14 double SVProcessStepData::mTopEggX

The top egg x factor.

4.8.3.15 double SVProcessStepData::mTopEggY

The top egg y factor.

4.8.3.16 SVProcessType SVProcessStepData::mType

The type of the process step.

Chapter 5

Interface Routines

This chapter describes the major interface routines. The routines are grouped in a namespace according to functionality. Each section describes a namespace and gives a list of its routines. A description of each routine is then provided.

5.1 SVInterface Namespace Reference

This class namespace routines to interface with the SUMMiTView Library.

Functions

- DD_RESULT **save_mems_geometry_to_file** (std::string filename, **SVMemsGeometry** *p_mems_geometry)
- DD_RESULT **restore_mems_geometry_from_file** (std::string filename, std::list< **SVMemsGeometry** * > &r_mems_geometries)
- std::string **get_version** ()
- void **create_process_steps** (std::list< **SVProcessStepData** * > &r_process_steps_data, std::map< std::string, **SVCompositeMask** * > &r_composite_masks, std::list< **SVProcessStep** * > &r_process_steps)
- std::string **get_process_name_from_type** (**SVProcessType** type)
- std::string **get_etch_type_name_from_type** (**SVEtchType** type)
- std::string **get_field_name_from_field** (**SVFieldType** field)
- void **get_layer_names_from_process_steps** (std::list< **SVProcessStepData** * > &r_process_steps_data, std::list< std::string > &r_used_layers)
- void **create_materials** (std::list< **SVProcessStepData** * > &r_process_steps_data)
- DD_RESULT **process_mask_file** (std::string &r_filename, double tolerance, DDMaskDefinitions &r_mask_definitions)
- DD_RESULT **process_mask_file** (std::string &r_filename, double tolerance, std::list< std::string > &r_layers, DDMaskDefinitions &r_mask_definitions)
- DD_RESULT **create_composite_masks** (std::list< **SVProcessStepData** * > &r_process_steps_data, double tolerance, DDMaskDefinitions &r_mask_definitions, std::map< std::string, **SVCompositeMask** * > &r_composite_masks)

5.1.1 Detailed Description

This class namespace routines to interface with the SUMMiTView Library.

Author:
Corey McBride

Date:
05/30/03

5.1.2 Function Documentation

5.1.2.1 DD_RESULT create_composite_masks (std::list< SVProcessStepData * > & *r_process_steps_data*, double *tolerance*, DDMaskDefinitions & *r_mask_definitions*, std::map< std::string, SVCompositeMask * > & *r_composite_masks*)

Creates the composite masks that are used in the process steps. This routine create compoiste masks by XORing the surfaces from one layer with the surface from another layer. Which layers to XOR together is determined by the individual process steps.

Parameters:

r_process_steps_data The list of **SVProcessStepData**(p. 85) that is used to determine how to create the masks. Each **SVProcessStepData**(p. 85) object represent one process step.

tolerance The tolerance used during the routine.

r_mask_definitions The SVMaskDefinition object that contains the surfaces for each layer indexed by layer name.

r_composite_masks The map containing the resulting SVCompositeMasks. Each **SVCompositeMask**(p. 56) is indexed by mask name.

5.1.2.2 void create_materials (std::list< SVProcessStepData * > & *r_process_steps_data*)

Creates the instances of **SVMaterial**(p. 38) to represent each material used by the process steps.

Parameters:

r_process_steps_data The **SVProcessStepData**(p. 85) objects to create the materials from.

5.1.2.3 void create_process_steps (std::list< SVProcessStepData * > & *r_process_steps_data*, std::map< std::string, SVCompositeMask * > & *r_composite_masks*, std::list< SVProcessStep * > & *r_process_steps*)

Creates a list of **SVProcessStep**(p. 61) instances based upon a list of **SVProcessStepData**(p. 85).

Parameters:

r_process_steps_data A list of **SVProcessStepData**(p. 85) objects used to create SVProcess-Steps.

r_composite_masks A map of SVCompositeMasks used in the process steps. The map is indexed by **SVCompositeMask**(p. 56) name.

r_process_steps The list of **SVProcessStep**(p. 61) instances that were created by the routine.

5.1.2.4 std::string get_etch_type_name_from_type (SVEtchType *type*)

Returns a std::string with the etch name based upon the SVEtchType.

Parameters:

type The SVEtchType to return the name of.

5.1.2.5 std::string get_field_name_from_field (SVFieldType *field*)

Returns a std::string with the field name based upon the SVFieldType.

Parameters:

field The SVFieldType to return the name of.

5.1.2.6 void get_layer_names_from_process_steps (std::list< SVProcessStepData * > & *r_process_steps_data*, std::list< std::string > & *r_used_layers*)

Gets a list of std::strings that contain the names of each layer used in the process steps.

Parameters:

r_process_steps_data The list of **SVProcessStepData**(p. 85) objects to get the layer names from.

r_used_layers The list of std::strings that contains the names of the layers used by the SVProcessStepDatas.

5.1.2.7 std::string get_process_name_from_type (SVProcessType *type*)

Returns a std::string with the process name based upon the SVProcessType.

Parameters:

type The SVProcessType to return the name of.

5.1.2.8 std::string get_version ()

Returns a string that contains the current library version information.

5.1.2.9 DD_RESULT process_mask_file (std::string & *r_filename*, double *tolerance*, std::list< std::string > & *r_layers*, DDMaskDefinitions & *r_mask_definitions*)

Creates DDSurfaces based upon the polygons in a .mem mask file. The DDSurfaces are stored in an instance of DDMaskDefinitions. The surfaces of each layer are unioned(OR) together.

Parameters:

r_filename The filename for the .mem file.

tolerance The tolerance used during the routine.

r_layers The list of layers to extract from the .mem file.

r_mask_definitions A DDMaskDefinitions object used to store the mask set.

5.1.2.10 DD_RESULT process_mask_file (std::string & *r_filename*, double *tolerance*, DDMaskDefinitions & *r_mask_definitions*)

Creates DDSurfaces based upon the polygons in a .mem mask file. The DDSurfaces are stored in an instance of DDMaskDefinitions. The surfaces of each layer are unioned(OR) together.

Parameters:

r_filename The filename for the .mem file.

tolerance The tolerance used during the routine.

r_mask_definitions A DDMaskDefinitions object used to store the mask set.

5.1.2.11 DD_RESULT restore_mems_geometry_from_file (std::string *filename*, std::list< SVMemsGeometry * > & *r_mems_geometries*)

Restores A **SVMemsGeometry**(p. 49) from a file.

Parameters:

filename The file to read.

r_mems_geometries The list of where the restored **SVMemsGeometry**(p. 49) pointers will be returned.

Return values:

DD_X_FILE_IO_ERROR If there was an error opening the file.

DD_FAILURE If there was an error restoring the geometry.

DD_SUCCESS If the restoring was successfull.

5.1.2.12 DD_RESULT save_mems_geometry_to_file (std::string *filename*, SVMemsGeometry * *p_mems_geometry*)

Saves A **SVMemsGeometry**(p. 49) to file.

Parameters:

filename The name of the file to create.

p_mems_geometry The **SVMemsGeometry**(p. 49) to save to file.

5.2 SVSatFileWriter Namespace Reference

Interface for writing 2 1/2d MEMS Geometry as 3d sat files.

Functions

- DD_RESULT **save_chunk_to_file_sat_format** (std::string &r_filename, DDSurface &r_surface, double height, double z_value)
- DD_RESULT **save_mems_geometry_to_file_sat_format** (std::string &r_filename, **SVMemsGeometry** &r_mems_geometry)
- DD_RESULT **export_sat_file_by_material** (std::string &r_filename, **SVMemsGeometry** &r_mems_geometry)
- DD_RESULT **export_sat_files_by_material** (std::string &r_root_filename, **SVMemsGeometry** &r_mems_geometry)
- DD_RESULT **export_sat_files_by_step** (std::string &r_root_filename, **SVMemsGeometry** &r_mems_geometry)
- DD_RESULT **export_sat_file_as_one_body** (std::string &r_filename, **SVMemsGeometry** &r_mems_geometry)

5.2.1 Detailed Description

Interface for writing 2 1/2d MEMS Geometry as 3d sat files.

Author:
Corey McBride

Date:
01/26/04

5.2.2 Function Documentation

5.2.2.1 DD_RESULT export_sat_file_as_one_body (std::string & *r_filename*, SVMemsGeometry & *r_mems_geometry*)

Exports a **SVMemsGeometry**(p. 49) to a SAT file. Exports all of the material as one merged body.

Parameters:
r_filename The name of the file to create.
r_mems_geometry The **SVMemsGeometry**(p. 49) to export.

5.2.2.2 DD_RESULT export_sat_file_by_material (std::string & *r_filename*, SVMemsGeometry & *r_mems_geometry*)

Exports a **SVMemsGeometry**(p. 49) to a SAT file. All of the chunks in the **SVMemsGeometry**(p. 49) of one material are exported as one body. All adjacent chunks of the same material at the same level are exported as one lump.

Parameters:

r_filename The name of the file to create.

r_mems_geometry The **SVMemsGeometry**(p. 49) to export.

5.2.2.3 DD_RESULT export_sat_files_by_material (std::string & *r_root_filename*, SVMemsGeometry & *r_mems_geometry*)

Exports a **SVMemsGeometry**(p. 49) to a series of sat files All chunks of one material are output in a .sat file, one file per material An index file named is also created.

Parameters:

r_filename The name of the file to create.

r_mems_geometry The **SVMemsGeometry**(p. 49) to export.

5.2.2.4 DD_RESULT export_sat_files_by_step (std::string & *r_root_filename*, SVMemsGeometry & *r_mems_geometry*)

Exports a **SVMemsGeometry**(p. 49) to a series of sat files All chunks of one step are output in a .sat file, one file per step An index file named is also created.

Parameters:

r_filename The name of the file to create.

r_mems_geometry The **SVMemsGeometry**(p. 49) to export.

5.2.2.5 DD_RESULT save_chunk_to_file_sat_format (std::string & *r_filename*, DDSurface & *r_surface*, double *height*, double *z_value*)

Exports a chunk to a SAT file. A chunk is a 3d geometry that is created by extruding a 2d DDSurface in the Z direction.

Parameters:

r_filename The name of the file to create.

r_surface The DDSurface to export.

height The Z value of the top of the 3d geometry.

z_value The distance to sweep the DDSurface in the negative Z direction.

5.2.2.6 DD_RESULT save_mems_geometry_to_file_sat_format (std::string & *r_filename*, SVMemsGeometry & *r_mems_geometry*)

Exports a **SVMemsGeometry**(p. 49) to a SAT file. Each individual chunk in the **SVMemsGeometry**(p. 49) is exported as a seperate body.

Parameters:

r_filename The name of the file to create.

r_mems_geometry The **SVMemsGeometry**(p. 49) to export.

5.3 SVParser Namespace Reference

This namespace contains the routines to parse the SUMMiTView process definition file.

Functions

- DD_RESULT **parse_process_definition_file** (std::string &r_filename, std::list< **SVProcessStepData** * > &r_process_step_data)
- DD_RESULT **get_path_parts** (std::string &r_file, std::string &r_path, std::string &r_name, std::string &r_extension)
- DD_RESULT **open_input_file** (std::string filename, std::ifstream &r_in_file)
- DD_RESULT **open_output_file** (std::string filename, std::ifstream &r_out_file)

5.3.1 Detailed Description

This namespace contains the routines to parse the SUMMiTView process definition file.

Author:
Corey McBride

Date:
06/22/03

5.3.2 Function Documentation

5.3.2.1 DD_RESULT get_path_parts (std::string & *r_file*, std::string & *r_path*, std::string & *r_name*, std::string & *r_extension*)

Separates a fully qualified file name into its parts.

Parameters:

r_file The std::string that contains the fully qualified file name.

r_path The std::string where the path is returned.

r_name The std::string where the file name is returned.

r_extension The std::static where the file extension is returned.

Return values:

DD_FAILURE If the file name was empty.

DD_SUCCESS If the routine was successfull.

5.3.2.2 DD_RESULT open_input_file (std::string *filename*, std::ifstream & *r_in_file*)

Opens a file for reading.

Parameters:

filename The name of the file to open.

r_in_file The variable where the open file is returned.

Return values:

DD_X_FILE_IO_ERROR If there was an error opening the file.

DD_SUCCESS If the routine was successfull.

5.3.2.3 DD_RESULT open_output_file (std::string *filename*, std::ifstream & *r_out_file*)

Opens a file for writing.

Parameters:

filename The name of the file to open.

r_out_file The variable where the open file is returned.

Return values:

DD_X_FILE_IO_ERROR If there was an error opening the file.

DD_SUCCESS If the routine was successfull.

5.3.2.4 DD_RESULT parse_process_definition_file (std::string & *r_filename*, std::list< SVProcessStepData * > & *r_process_step_data*)

Parses the SUMMiTView process definition file. The data from the process definition file is stored in a list of **SVProcessStepData**(p. 85) objects. There is one instance of **SVProcessStepData**(p. 85) for each process step.

Parameters:

r_filename The name of the process definition file.

r_process_step_data The list where the new **SVProcessStepData**(p. 85) objects are returned.

Return values:

DD_X_FILE_IO_ERROR If there was an error opening the process definition file.

DD_FAILURE If an error unrecognized command was encountered in the file.

DD_SUCCESS If the routine was successfull.

5.4 SVModifyGeometryTool Namespace Reference

This namespace contains general utiliy routines to modify SVChunks, SVLevels and SVMemsGeometrys.

Functions

- DD_RESULT **reverse_composite_mask** (**SVCompositeMask** &r_composite_mask, double tolerance, DDBoundingBox &r_bounding_box)
- void **split_level** (**SVLevel** &r_level, double top_z_value, double bottom_z_value, **SVLevel** *&rp_level)
- void **remove_chunks** (DDSurface *p_surface, **SVMemsGeometry** &r_mems_geometry)
- DD_RESULT **split_chunks** (DDSurface *p_surface, **SVMemsGeometry** &r_mems_geometry, std::list< DDSurface * > &r_surface_list)
- DD_RESULT **add_chunks** (DDSurface *p_surface, **SVMemsGeometry** &r_mems_geometry, std::list< DDSurface * > &r_surface_list)
- DD_RESULT **add_level** (**SVMemsGeometry** &r_mems_geometry, SVMemsGeometry::iterator &r_pos, double z_value, **SVMaterial** *p_material, **SVLevel** *&rp_level)
- DD_RESULT **find_all_reachable_chunk_clusters** (**SVMemsGeometry** &r_mems_geometry, SVMemsGeometry::iterator &r_level_iterator, std::list< **SVMaterial** * > &r_affected_materials, DDHashMap< int, std::list< **SVMGTChunkCluster** * > * > &r_reachable_chunks)
- DD_RESULT **find_merge_chunks** (**SVMemsGeometry** &r_mems_geometry, std::list< **SVMGTChunkCluster** * > &r_cluster_list)
- DD_RESULT **find_merge_chunks** (**SVLevel** &r_level, **SVMemsGeometry** &r_mems_geometry, DDHashMap< int, std::list< DDSurface * > * > &r_surface_neighbor_map, std::list< **SVMGTChunkCluster** * > &r_cluster_list)
- DD_RESULT **merge_levels** (**SVMemsGeometry** &r_mems_geometry)
- DD_RESULT **merge_adjacent_chunks** (**SVMemsGeometry** &r_mems_geometry)
- void **remove_all_void_levels_from_top** (**SVMemsGeometry** &r_mems_geometry, **SVMaterial** *p_void_material)

5.4.1 Detailed Description

This namespace contains general utiliy routines to modify SVChunks, SVLevels and SVMemsGeometrys.

Author:
Corey McBride

Date:
06/02/03

5.4.2 Function Documentation

5.4.2.1 DD_RESULT add_chunks (DDSurface * *p_surface*, SVMemsGeometry & *r_mems_geometry*, std::list< DDSurface * > & *r_surface_list*)

Adds new SVChunks to the **SVMemsGeometry**(p. 49). The new SVChunks are created to have the same **SVMaterial**(p. 38) at each **SVLevel**(p. 44) as the SVChunks for the DDSurface provided.

Parameters:

p_surface The DDSurface for the SVChunks to copy the **SVMaterial**(p. 38) from.

r_mems_geometry The **SVMemsGeometry**(p. 49) to add the SVChunks to.

r_surface_list The DDSurfaces to add new SVChunks for.

Return values:

DD_X_CRITICAL_ERROR If the routine experienced a critical error.

DD_SUCCESS If the routine was a success.

5.4.2.2 DD_RESULT add_level (SVMemsGeometry & *r_mems_geometry*, SVMemsGeometry::iterator & *r_pos*, double *z_value*, SVMaterial * *p_material*, SVLevel *& *rp_level*)

Adds a new **SVLevel**(p. 44) to the **SVMemsGeometry**(p. 49). The new **SVLevel**(p. 44) will has SVChunks for each DDSurface of the **SVMemsGeometry**(p. 49). The new SVChunks will be of the **SVMaterial**(p. 38) provided.

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to add the new **SVLevel**(p. 44) too.

r_pos The **SVMemsGeometry**(p. 49) iterator to add the new **SVLevel**(p. 44) before.

z_value The z value for the new **SVLevel**(p. 44).

p_material The **SVMaterial**(p. 38) for the SVChunks of the new **SVLevel**(p. 44).

rp_level The variable where the new **SVLevel**(p. 44) will be returned.

Return values:

DD_X_NULL_POINTER If the p_material pointer is NULL;

DD_SUCCESS If the routine was successfull.

5.4.2.3 DD_RESULT find_all_reachable_chunk_clusters (SVMemsGeometry & *r_mems_geometry*, SVMemsGeometry::iterator & *r_level_iterator*, std::list< SVMaterial * > & *r_affected_materials*, DDHashMap< int, std::list< SVMGTChunkCluster * > * > & *r_reachable_chunks*)

Finds all SVChunks of affected **SVMaterial**(p. 38) that are reachable from the provided **SVLevel**(p. 44) of the **SVMemsGeometry**(p. 49). SVChunks are considered reachable if they are of they affected SVMaterials and a path can be traced from the **SVChunk**(p. 37) to the provided **SVLevel**(p. 44). The path can only pass through SVChunks of affected SVMaterials. The reachable SVChunks are grouped as adjacent SVChunks called **SVMGTChunkCluster**(p. 59). The **SVMGTChunkCluster**(p. 59) are returned indexed by the **SVLevel**(p. 44) where they reside. Thus each **SVMGTChunkCluster**(p. 59) is only spans one **SVLevel**(p. 44).

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to find the reachable SVChunks from.

r_level_iterator The **SVLevel**(p. 44) to start checking for reachable SVChunks. SVChunks above this **SVLevel**(p. 44) are not considered.

r_affected_materials The SVMaterials that are considered affected **SVMaterial**(p. 38).

r_reachable_chunks The DDHashMap that contains pointers to the reachable SVChunks grouped by **SVMGTChunkCluster**(p. 59) and indexed by **SVLevel**(p. 44).

Return values:

DD_FAILURE If no affected **SVMaterial**(p. 38) were provided. Or if the **SVLevel**(p. 44) iterator is not valid.

DD_SUCCESS If the routine was successfull.

5.4.2.4 DD_RESULT find_merge_chunks (SVLevel & *r_level*, SVMemsGeometry & *r_mems_geometry*, DDHashMap< int, std::list< DDSurface * > * > & *r_surface_neighbor_map*, std::list< SVMGTChunkCluster * > & *r_cluster_list*)

Finds all mergeable SVChunks in a **SVLevel**(p. 44). Mergeable SVChunks are those SVChunks that come from adjacent DDSurfaces and are of the same material. The mergable SVChunks are returned as groups of adjacent SVChunks called SVMGTChunkClusters. The SVMGTChunkClusters only span one **SVLevel**(p. 44) of the **SVMemsGeometry**(p. 49).

Parameters:

r_level The **SVLevel**(p. 44) to find the mergeable SVChunks on.

r_mems_geometry The **SVMemsGeometry**(p. 49) where the **SVLevel**(p. 44) resides.

r_surface_neighbor_map A map containing the list of neighboring DDSurfaces indexed by DDSurface. This map may start empty and will be filled in by the routine. Or the map may contain information from a previous execution of this routine.

r_cluster_list The list where the SVMGTChunkClusters that contain the mergable SVChunks are returned.

Return values:

DD_X_CRITICAL_ERROR If the routine experienced a critical error.

DD_SUCCESS If the routine was a success.

5.4.2.5 DD_RESULT find_merge_chunks (SVMemsGeometry & *r_mems_geometry*, std::list< SVMGTChunkCluster * > & *r_cluster_list*)

Finds all mergeable SVChunks in a **SVMemsGeometry**(p. 49). Mergeable SVChunks are those SVChunks that come from adjacent DDSurfaces and are of the same material. The mergable SVChunks are returned as groups of adjacent SVChunks called SVMGTChunkClusters. The SVMGTChunkClusters only span one **SVLevel**(p. 44) of the **SVMemsGeometry**(p. 49).

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to find all mergable **SVChunk**(p. 37) in.

r_cluster_list The list where the SVMGTChunkClusters that contain the mergable SVChunks are returned.

Return values:

DD_X_CRITICAL_ERROR If the routine experienced a critical error.

DD_SUCCESS If the routine was a success.

5.4.2.6 DD_RESULT merge_adjacent_chunks (SVMemsGeometry & *r_mems_geometry*)

Merges adjacent SVChunks. DDSurfaces and their corresponding SVChunks are merged if the DDSurfaces are adjacent to each other and if their SVChunks have the same material are each **SVLevel**(p. 44) in the **SVMemsGeometry**(p. 49).

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to merge the SVChunks in.

Return values:

DD_X_CRITICAL_ERROR If the routine experienced a critical error.

DD_SUCCESS If the routine was a success.

5.4.2.7 DD_RESULT merge_levels (SVMemsGeometry & *r_mems_geometry*)

Merges SVLeves that are adjacent in the **SVMemsGeometry**(p. 49) and have SVChunks of the same **SVMaterial**(p. 38) for each DDSurface.

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to merge the SVLevels in.

Return values:

DD_X_CRITICAL_ERROR If the routine experienced a critical error.

DD_SUCCESS If the routine was a success.

5.4.2.8 void remove_all_void_levels_from_top (SVMemsGeometry & *r_mems_geometry*, SVMaterial * *p_void_material*)

Removes all void SVLevels from the top of a **SVMemsGeometry**(p. 49). A **SVLevel**(p. 44) is considered void if all of the SVChunks in the **SVLevel**(p. 44) are of the void **SVMaterial**(p. 38). The top most **SVLevel**(p. 44) is considered first. Once a non void **SVLevel**(p. 44) is encountered the routine returns.

Parameters:

r_mems_geometry The **SVMemsGeometry**(p. 49) to remove void SVLevels from.

p_void_material The **SVMaterial**(p. 38) to consider as void **SVMaterial**(p. 38).

5.4.2.9 void remove_chunks (DDSurface * *p_surface*, SVMemsGeometry & *r_mems_geometry*)

Removes all of the SVChunks from a **SVMemsGeometry**(p. 49) for a DDSurface.

Parameters:

p_surface The DDSurface for the SVChunks to remove.

r_mems_geometry The **SVMemsGeometry**(p. 49) to remove the SVChunks from.

5.4.2.10 DD_RESULT reverse_composite_mask (SVCompositeMask & *r_composite_mask*, double *tolerance*, DDBoundingBox & *r_bounding_box*)

Reverses a **SVCompositeMask**(p. 56). The **SVCompositeMask**(p. 56) is reversed by creating DDSurfaces to cover the areas that are holes in the original DDSurfaces. The area that is originally covered by DDSurfaces becomes holes in the new DDSurfaces.

Parameters:

r_composite_mask The **SVCompositeMask**(p. 56) to reverse.

tolerance The tolerance used during the routine.

r_bounding_box The DDBoundingBox that defined the bounding box of the original DDSurfaces.

Return values:

DD_X_CRITICAL_ERROR If the routine experienced a critical error.

DD_SUCCESS If the routine was a success.

5.4.2.11 DD_RESULT split_chunks (DDSurface * *p_surface*, SVMemsGeometry & *r_mems_geometry*, std::list< DDSurface * > & *r_surface_list*)

Splits all of the SVChunks in a **SVMemsGeometry**(p. 49) for a DDSurface. All of the **SVChunk**(p. 37) for a DDSurface are removed and new SVChunks are added for the new DDSurfaces. The new **SVChunk**(p. 37) have the same **SVMaterial**(p. 38) as the original **SVChunk**(p. 37). The new DDSurfaces must replace the original DDSurface in the **SVMemsGeometry**(p. 49). This means that the new DDSurfaces must be interconnected and tile the area covered by the original DDSurface. In addition the new DDSurfaces must be interconnected with the DDSurfaces that were neighbors of the original DDSurface.

Parameters:

p_surface The DDSurface for the SVChunks to split.

r_mems_geometry The **SVMemsGeometry**(p. 49) to split the SVChunks in.

r_surface_list The new DDSurfaces that replace the original DDSurface.

Return values:

DD_X_CRITICAL_ERROR If the routine experienced a critical error.

DD_SUCCESS If the routine was a success.

5.4.2.12 void split_level (SVLevel & *r_level*, double *top_z_value*, double *bottom_z_value*, SVLevel *& *rp_level*)

Splits a **SVLevel**(p. 44) into two SVLevels. The **SVLevel**(p. 44) is split by creating a new **SVLevel**(p. 44) and then assiging new z values to each instance. If the original **SVLevel**(p. 44) belonged to a **SVMemsGeometry**(p. 49) then the new **SVLevel**(p. 44) is then inserted after the original **SVLevel**(p. 44).

Parameters:

r_level The **SVLevel**(p. 44) to split.

top_z_value The new z value for the original **SVLevel**(p. 44).

bottom_z_value The new z value for the new **SVLevel**(p. 44).

rp_level The new **SVLevel**(p. 44) that is created.

5.5 SVSummitViewApp Namespace Reference

This class contains routines that use the SUMMiTView Library to create 2 1/2d geometry.

Functions

- DD_RESULT **execute** (std::string &r_mask_file, std::string &r_process_file, std::string &r_log_file, double tolerance, double mask_tolerance, **SVMemsGeometry** *&rp_mems_geometry, bool b_save_after_each_step=false)

5.5.1 Detailed Description

This class contains routines that use the SUMMiTView Library to create 2 1/2d geometry.

Author:
Corey McBride

Date:
05/30/03

5.5.2 Function Documentation

5.5.2.1 DD_RESULT execute (std::string & *r_mask_file*, std::string & *r_process_file*, std::string & *r_log_file*, double *tolerance*, double *mask_tolerance*, SVMemsGeometry *& *rp_mems_geometry*, bool *b_save_after_each_step* = `false`)

This routine uses the SUMMiTView Library to create 2 1/2d geometry from a process file and a 2d mask set.

Parameters:

r_mask_file The file name of the 2d mask set.

r_process_file The file name of the process file.

r_log_file The file name of the log file to create.

tolerance The tolerance used to create the 2 1/2d geometry.

mask_tolerance The tolerance used when processing the 2d mask set.

rp_mems_geometry The **SVMemsGeometry**(p. 49) created by the routine.

b_save_after_each_step The flag to indicate if the geometry is to be saved after each process step. The flag defaults to false.

References

[1] C. R. Jorgensen and V. Yarberry. ”A 3D geometry modeler for the SUMMiT V MEMS designer”. In *Technical Proceedings of the 2001 International Conference on Modeling and Simulation of Microsystems*, volume 1, pages 594 – 597, 2001.

[2] mems.sandia.gov/samples/doc/Process_Definition_File_Documentation6.pdf.

[3] mems.sandia.gov/samples/doc/MEM_File_Format_Documentation.txt.

[4] C. L. McBride, V. Yarberry, and R. C. Schmidt. GBL-2D Version 1.0: A 2D Geometry Boolean Library. Technical report SAND2006-6829, Sandia National Laboratories, P.O. Box 5800, Albuquerque, New Mexico 87185, 2006.

[5] M. S. Rodgers and J. J. Sniegowski. ”Designing Microelectromechanical Systems-on-a-Chip in a 5-Level Surface Micromachine Technology”. In *2nd International Conference on Engineering Design and Automation*, 1998.

[6] V. R. Yarberry. ”Meeting the MEMS ’design-to-analysis’ challenge: the SUMMiT V design tool environment”. In *Presented at the ASME International Mechanical Engineering Congress*, New Orleans, LA, November 17-22 2002.

[7] V. Yarberry and C. Jorgensen. A 2D visualization tool for SUMMiT V design. Technical report SAND2000-3126, Sandia National Laboratories, P.O. Box 5800, Albuquerque, New Mexico 87185, 2000.

[8] V. R. Yarberry. ”A 2D visualization tool for SUMMiT V designs”. In *Proceedings of the Fourth International Conference on Modeling and Simulation of Microsystems*, pages 606–609, Hilton Head Island, CS, March 19-21 2001.

[9] J. Allen. ”MEMS design rule checking: A batch approach for remote operation”. In *Proceedings of the SPIE 5th Annual International Symposium on Smart Electronics and MEMS*, volume 3328, pages 32–39, San Diego, CA, March 1-5 1998.

[10] Spatial Corp. ACIS R12 Online Help. Library documentation, Spatial Corp., A Dassault Systemes company, 10955 Westmoor Drive, Suite 425 Westminster, Colorado 80021, 2003.

[11] www.stlport.org.

Appendix 1: Example Code that uses SummitView

The example C++ code listed here creates a 2 1/2D geometry from a MEMS mask file and a process definition file.

```
#include <string>
#include <fstream>
#include <list>

#include "SVParser.hpp"
#include "SVMemsGeometry.hpp"
#include "SVModifyGeometryTool.hpp"
#include "DDMaskDefinitions.hpp"
#include "DDModifyGeometryTool.hpp"
#include "SVInterface.hpp"
#include "DDSurface.hpp"
#include "SVMaterial.hpp"
#include "SVWeaver.hpp"
#include "DDInterface.hpp"
#include "SVSatFileWriter.hpp"

void main(int argc, char* argv[])
{
  // This example creates a 2 1/2D geometry from a MEMS mask file and
  // a process definition file.

  // Parse the process definition file.
  // This produces a list of process step data.
  std::cout<<"-Reading Process Definition File.\n";

  std::string process_file_name;
  process_file_name="process_def.txt";
  std::list<SVProcessStepData*> process_steps_data;

  DD_RESULT result =  SVParser::parse_process_definition_file(
    process_file_name,
    process_steps_data);
  if(result != DD_SUCCESS)
  {
    std::cout<<"-Error Reading Process Definition File.\n";
    return ;
  }

  // Sort the process step data based on process step number.
  // The is so that the steps will be processed in the correct order.
  // This call uses the std::list sort routine with a custom sort function.
  process_steps_data.sort(SVProcessStepDataLessThan());

  // Parse the MEMS file.
  std::cout<<"-Processing MEMS File.\n";

  // This gets the layers of the MEMS file that are used.
  // This saves time by processing only those masked that are specified
```

```
// in the process definition file.
std::list<std::string> used_layers;
SVInterface::get_layer_names_from_process_steps(
  process_steps_data,
  used_layers);

// This is the MEMS file to process.
std::string mask_file_name;
mask_file_name="example.mem";

// This creates the GBL-2D Geometry representation for the MEM masks.
// The geometry is saved in a DDMaskDefinitions object.
// This object saves the geometry in lists. There is one list for each
// layer in the MEMS file.
DDMaskDefinitions mask_definitions;
result = SVInterface::process_mask_file(
  mask_file_name,
  .001,
  used_layers,
  mask_definitions);
if(result == DD_X_CRITICAL_ERROR)
{
 std::cout<<"-Error Processing MEMS Mask File.\n";
 return ;
}

// This prints the MEMS file layers that have been processed.
// This is also the layers that are used by the process definition file.
mask_definitions.print_information();

// This calculates the total bounding box for the MEMS Masks that are used.
// This will be used later to set up the initial 2 1/2D geometry.
DDBoundingBox box;
box=mask_definitions.get_bounding_box();

// This creates the composite masks.
// The composite masks are the masks that are used by the individual process steps.
// They are created by XORing different MEMS file layers together.
// These layers are specified as Masks and Cut Masks with the
// same process step name in the process definition file.
std::cout<<"-Creating Composite Masks.\n";
std::map<std::string, SVCompositeMask*> composite_masks;
result = SVInterface::create_composite_masks(
  process_steps_data,
  .001,
  mask_definitions,
  composite_masks);
if(result == DD_X_CRITICAL_ERROR)
{
 std::cout<<"-ERROR Creating Composite Masks.\n";
 return;
}

// This creates the materials that are specified in the process definition file.
```

```
std::cout<<"-Creating Materials.\n";
SVInterface::create_materials(process_steps_data);

// This create the process steps.
// A process step of the correct type is created with all of the necessary information to
// perform desired process.
// For example, an instance of SVDryEtch is created for a Dry Etch step.
// The etch depth, affected materials and etch mask are assigned to the instance.
// All process steps are derived from SVProcessStep.
// Thus the list of SVProcessSteps contain all of the process steps specified in
// process definition file.
std::cout<<"-Creating Process Steps.\n";
std::list<SVProcessStep*> process_steps;
SVInterface::create_process_steps(
    process_steps_data,
    composite_masks,
    process_steps);


// This creates the initial 2 1/2D geometry.
std::cout<<"-Creating initial geometry.\n";
SVMemsGeometry* p_mems_geometry = new SVMemsGeometry(box);
p_mems_geometry->set_tolerance(.001);

// A surface is created from the bounding box of the masks.
// This surface defines the working area where the geometry will be created.
// Because the geometry is created by simulating the geometric effects of various
// process steps, it is necessary to have a working area within which the process steps
// can operate.
// The working area needs to be sufficiently large to encompass the final 2 1/2D geometry.
DDSurface* p_surface;
DDModifyGeometryTool::create_surface_from_bounding_box(box,p_surface);
p_mems_geometry->mSurfaceList.push_back(p_surface);

// This sorts the process steps based the process step number. This is so that
// the process steps will be processed in the correct order.
process_steps.sort(SVProcessStepLessThan());

// This performs each process step on the working volume.
// The process steps are executed in the order that the appear in the list.
std::cout<<"-Creating 2 1/2D MEMS Geometry.\n";
std::list<SVProcessStep*>::iterator i,i_end;
i=process_steps.begin();
i_end=process_steps.end();
for(;i!=i_end;i++)
{
  // This prints out the information for the process step.
  (*i)->print_information();

  DD_RESULT result =  (*i)->execute(*p_mems_geometry);
  if(result == DD_X_CRITICAL_ERROR)
  {
   std::cout<<"-Error Creating Geometry.\n";
   return;
```

```
    }

    // This merges levels that share the same geometry.
    // This is a refinement that increases speed and performance.
    SVModifyGeometryTool::merge_levels(*p_mems_geometry);
  }

 // This saves the created 2 1/2D MEMS Geometry to a ACIS SAT file.
  std::cout<<"\n-Finished Creating 2 1/2D MEMS Geometry.\n\n";
  std::cout << "Saving MEMS 2 1/2D Geometry to SAT File by Material";
  std::string output_filename;
  output_filename="Example.sat";
  SVSatFileWriter::export_sat_files_by_material(output_filename,*p_mems_geometry);

  std::cout << "Program Finished\n";
}
```

DISTRIBUTION:

5 Corey L. McBride
Elemental Technologies
17 North Merchant Street
American Fork, UT 84003

1 Ray Myers
Elemental Technologies
17 North Merchant Street
American Fork, UT 84003

5 MS 0316
Rodney C. Schmidt, 1437

1 MS 0316
Richard Schiek, 1437

1 MS 0316
Scott Hutchinson, 1437

1 MS 0321
Jennifer Nelson, 1430

5 MS 1069
Vic Yarberry, 1737

1 MS 1080
Dave Sandison, 1749

1 MS 1243
Steve Kempka, 5535

2 MS 9018
Central Technical Files, 8944

2 MS 0899
Technical Library, 4536