Int J Adv Manuf Technol (2001) 17:453–461
2001 Springer-Verlag London Limited

The International Journal of
Advanced
Manufacturing
Technology

Three-Dimensional Kernel Development for Injection Mould Design

T. L. Neo and K. S. Lee

Department of Mechanical and Production Engineering, National University of Singapore, Singapore

Today, many software "plug-ins" have been developed on high-level 3D modelling platforms to facilitate processes such as FEM analysis, CAM, injection mould design, simulation and visualisation. Such an arrangement is advantageous in many ways. However, it is not without shortcomings. Ideally, these "plug-ins" could also be developed using low-level 3D kernels for higher flexibility and better portability. This paper examines the various issues and methodologies related to the development of such 3D-based applications. The emphasis is placed on the software aspect. First, a methodology for the development of 3D-based applications is proposed. The idea is then implemented by developing an injection mould design application using a low-level 3D kernel called Parasolid. Based on design concepts used in an established mould design application, IMOLD, the development of a mould base design module is illustrated. An object-oriented programming language has been chosen for the development of the software on a Windows NT platform.

Keywords: 3D kernel; Computer-aided design (CAD); Injection mould design; Parasolid

1. Introduction

Three-dimensional CAD systems have increasingly been used to speed up the product realisation process. One of the first steps involved in the automation of the product design process is the creation of the component parts in a 3D modelling application. The 3D model, upon creation, is called the digital master copy. This 3D digital model forms the key to a wide spectrum of process automation.

Creating the 3D digital model of component parts is only the very first step. There are several other secondary tasks that must to be done before the part can be manufactured. Such tasks include finite-element analysis, jigs and fixtures design, injection mould design, computer-aided manufacturing, simulation, and visualisation. Today, many application Plug-ins have been developed on high-level 3D modelling platforms to facilitate these secondary tasks. The 3D-modelling platform provides the plug-in software with a library of functions as well as an established user interface and style of programming. As a result, the development times for these plug-ins are significantly reduced.

Such an arrangement is advantageous in many ways. However, it has its shortcomings, especially in the long run. In order to develop a plug-in for established software, the developers must adhere to the many constraints imposed. There is a need to be consistent with the style of the parent software. The developers must be able to achieve any functionality they need with only the set of library functions provided. Most end-users need both the parent software and the plug-in. In many cases, however, they may be more interested in using only the plug-in software. An example of such a situation is in injection mould design. These users, however, must purchase the entire software package which includes many features and functions that they do not need. Such a large program is often very demanding on the hardware, which also means higher cost. The plug-in software is also very dependent on developments in the parent software. Whenever a new version is updated for the parent software, the plug-in developers have to follow-up on the changes. These shortcomings may not exist if these applications were developed on a low-level platform. Ideally, these plug-ins could be developed using low-level 3D kernels for higher flexibility and better portability. In many instances, such a move is both feasible and advantageous.

Traditionally, injection mould design is carried out directly on a CAD system. The entire injection mould, consisting of perhaps hundreds of components, is modelled and assembled on CAD systems such as AutoCAD, Pro/Engineer, and Unigraphics. As the injection mould design process is recursive, it is very time-consuming to re-model and re-assemble the design. In this aspect, 3D CAD systems such as Pro/Engineer and Unigraphics, which are feature-based, have a significant advantage over 2D CAD systems such as AutoCAD. To further speed up the injection mould design process, plug-ins were developed on these 3D systems to automate certain stages of the design process. Examples of such add-on applications include IMOLD (Intelligent Mold Design and Assembly Sys-

Correspondence and ofprint requests to: K.-S. Lee, Department of Mechanical and Production Engineering, National University of Singapore, 119260 Singapore. E-mail: mpeleeksKnus.edu.sg

tem, developed at the National University of Singapore, based on Unigraphics), Expert Mold Designer (based on CADKEY) and Moldmaker (based on EUCLID). As each is based on a specific CAD system, there is no plug compatibility.

In 1994, Mok and Cheung [1] presented work on the development of an injection mould design application based on Unigraphics. In 1997, Shah [2] proposed a 3-tier architecture for standardising communications between geometric modelling kernels and applications that require geometric modelling services. His objective is to achieve plug compatibility between 3D applications that are based on Parasolid [3] (a 3D kernel, developed at the University of Cambridge) and ACIS. This, however, involved an extensively developed 3-tier modelling husk. In this paper, the author attempts to develop a lightweight injection mould design application using a low-level 3D kernel directly. The focus is on the flexibility and speed of the software development. Design concepts and procedures were taken from IMOLD [4,5], a complete mould design and assembly 3D application. Although the discussion is limited to injection mould design only, the methodology applied can easily be applied in other 3D-based applications that are of a similar nature.

A combination of developer tools was chosen for this purpose. Before the methodology is discussed, brief introductions to some of these tools are first presented. They are, IMOLD, Parasolid version 10.1, Visual C version 6.0, and the Microsoft Foundation Classes.

2. IMOLD as a Mould Design Application

IMOLD (Intelligent Mold Design and Assembly) is an established 3D-based application that is dedicated to injection mould design. It is developed on top of an advanced CAD system called Unigraphics. The development is carried out using the applications programming interface (API) provided. The software enables mould designers to create a design rapidly by providing the tools that are commonly needed. Standard components parts, that are often required in the design, have been pre-created in the software and can be readily used by the designer. This reduces the design time significantly. The mould design process is divided into several stages, providing the designer with a consistent method of creating the mould design. They are, namely:

1. Data preparation.
2. Filling system design.
3. Mould base design.
4. Inserts and parting design.
5. Cooling system design.
6. Slider and lifter design.
7. Ejection system design.
8. Standard parts library.

Each stage can be considered as an independent module of the program. The 3D-based requirements for each module vary only slightly. The success in developing the mould base module implies feasibility in developing all the other modules.

3. Parasolid as a 3D Kernel

Parasolid is designed to be the centre or “kernel” of any system that is based on 3D model data. It is essentially a solid modeller, which can be used to:

1. Build and manipulate solid objects.
2. Calculate mass and moments of inertia, and perform clash detection.
3. Output the objects in various ways, including pictorially.
4. Store the objects in some sort of database or archive, and retrieve them later.

Parasolid is one of the most advanced 3D kernels among CAD applications. It is the 3D kernel of Unigraphics and Solid-Works. Its unique tolerant modelling functionality enables it to accept data stored in other modeller formats. Parasolid model files are thus very potable. It is, therefore, a superior platform for the development of stand-alone applications.

The 3D-based application interacts with Parasolid through one of its three interfaces (see Fig. 1). These are called the Parasolid kernel (PK) interface, the kernel interface (KI) and the downward interface. The PK interface and the kernel interface sit “on top” of the modeller (side-by-side), and are the means by which the application models and manipulates the objects, as well as controls the functioning of the modeller. The downward interface lies “beneath” the modeller, and is called by the modeller when it needs to perform data-intensive or system type operations. It consists of three parts: frustrum; graphical output (GO); and foreign geometry. These are briefly explained below.

3.1 The KI and PK Interface

The KI and the PK are interfaces for the programmer to access the modelling capabilities in the Parasolid kernel. They are standard libraries of modelling functions. The programmer calls these modelling functions in their programs. As the KI is to be phased out soon, we chose to use the PK interface.

Fig. 1. Parasolid components.

3.2 The Frustrum

The frustrum is a set of functions, which must be written by the applications programmer. The kernel calls them when data must be saved or retrieved. When using Parasolid, the applications programmer must first decide how to manage the storage of data, which Parasolid outputs through the frustrum. Transferring data through the frustrum usually involves writing to, or reading from, files. The format and location of the files is determined when writing the frustrum functions.

3.3 The Graphical Output (GO)

The graphical output is another set of functions, which is to be written by the applications programmer. When a call is made to the PK rendering functions, the graphical data generated are output through the GO interface. The graphical data are then passed to a 3D rendering package. OpenGL, a software interface to graphic cards, is a rendering package that is used for our purpose.

3.4 The Foreign Geometry

The foreign geometry provides functionality for the development of customised geometrical types such as in-house curves and surfaces. These are used together with the standard geometrical types for modelling within Parasolid.

4. Object-Oriented Programming Using Visual C++ and the Microsoft Foundation Classes

Object-oriented programming (OOP) has been the undisputed option for software developers. It is among the most advanced developmental tools available. The Microsoft Visual Studio is such a software package. It features several developmental tools that are meant for Internet-based and Windows-based programming. Among these tools are the Visual C++ (VC++) and the Microsoft Foundation Classes (MFC). The VC++ is a powerful development tool for object-oriented programming, whereas the MFC is a framework of C++ classes that are dedicated to Windows-based programming. Together, these provided the applications programmer with powerful development features and functionalities such as auto-code generation, and wizard-based operations. These greatly improved productivity. The entire user-interface for our program is developed using the VC++ and the MFC.

5. System Design

The direct development of a 3D-based add-on application using a 3D kernel requires several issues to be addressed. They consist of 3 main stages at the highest level. First, the identification of the crucial features and functions required for the plug-in application. Secondly, the development of the design for the application framework. Lastly, the design and development of the individual modules in the framework with appropriate developmental tools.

5.1 Identification of Essential Modules

Parasolid, as a 3D kernel, provides only a number of libraries and a conceptual framework for 3D application development. It is thus necessary for the developers to identify and develop the other essential facilities that are provided in a 3D CAD system. In order to identify the required facilities, it is important to understand the discrepancies between the two. Table 1 summarises the main differences in the facilities provided by a 3D kernel and a 3D CAD system. Some of these facilities, such as features and parametric modelling, are both time-consuming and technically demanding to develop. As most plug-ins do not use all the facilities of the parent software, it is possible to develop only those required by the plug-ins using low-level 3D kernels, producing a standalone version.

Items 7 to 9 in Table 1 are prerequisites for the development of 3D-based applications using Parasolid. By studying the requirements of the plug-in application, other essential facilities can be identified. A framework for the application is then proposed, based on the facilities provided by the Parasolid kernel.

5.2 Framework for 3D-Based Applications

A framework is developed with reference to the facilities provided by the developmental tools and the requirements of the application. It is designed so that there are minimum dependencies between individual code modules. This may result in a small degree of code duplication. In exchange, there is better portability of the program codes, greater ease of maintenance and a better prospect for future expansion. The overview of this framework is illustrated in Fig. 2. The details of the various modules are discussed in the following sections.

5.2.1 Windows-Based User-Interface (A)

Parasolid does not provide the programmer with a userinterface. Thus, the development of the 3D-based application at every single stage will involve designing the user-interface from scratch. The necessary developments involve:

1. Environmental setting and display of the 3D-based application.
2. Interactive graphical interface and execution procedure for all application functionality.

5.2.2 3D Developer Layer (B)

Since different 3D-based applications require 3D-facilities to different extent, the framework must provide for these variations. A 3D developer layer (See Fig. 2, Item B) is conceptualised to handle such variations. It is a library of objects or classes that are developed, based on the Parasolid kernel. The extent of development depends on the requirements of the

Table 1. Summary of facilities provided by a 3D kernel and a CAD system.

Facilities	3D kernel	3D CAD system
1. Basic 3D modelling	Low-level and general functions provided	High-level and specific functions provided
2. Assemblies	Several library functions provided	Complete system provided
3. Feature-based modelling	Not provided	Established feature set provided
4. Parametric modelling	Not provided	Often provided
5. Free-form modelling	Low-level functions provided	Often provided
6. Drafting	Not provided	Complete system provided
7. Interactive user-interface	Not provided	Always provided
8. Visualisation of 3D objects	Conceptual framework and several library functions provided	Completely developed
9. File management system	Conceptual framework and several library functions provided	Completely developed

Fig. 2. Overview of 3D-based application.

application identified in the previous section. Besides catering for variations in application requirements, the 3D developer layer also acts as a programming interface for non-Parasolid developers. Such an interface can also be re-used for subsequent development of other 3D-based applications. The 3D developer layer essentially consists of three main sections. They are used for 3D modelling and assembly, 3D visualisation and 3D data management, respectively.

I. 3D Modeling and Assembly. The 3D modelling and assembly module is the most important and elaborate of all three sections. It is analogous to the application-programming interface (API) provided by most CAD systems. The module consists of a library of 3D-based objects or classes, which are used for the development of the core application modules. The basic 3D functionality that is required by most 3D applications must be developed first. Depending on the requirements of the individual 3D-based application, other more advance features are subsequently added.

II. *3D Visualisation.* The display of 3D objects in a Windows client area requires a software graphics interface. The graphical output together with a selected graphical interface, are used for the rendering of 3D objects in the 3D-based application, as well as the management of the viewing projections and transformations. Here, a library of classes is developed for such purposes.

III. *3D Data Management.* The 3D data management module is developed on top of the frustrum. The frustrum is the module in the Parasolid kernel that facilitates archiving and access of 3D part files. A library of classes are developed using the frustrum for handling:

1. 3D object file format.
2. File management operations such as opening and saving a 3D object file.

5.2.3 Application Modules (C)

These are the actual 3D-based application modules that sit between the 3D developer layer and the application userinterface. The design of these modules depends mainly on the nature of the applications and often differs greatly. The main bulk of the developmental work is carried out in this area. The ease of the development, however, depends on the capabilities of the 3D developer layer.

5.2.4 Other Software Modules (D)

Very often, the 3D-based application may require functionality from other existing software modules or application modules. Therefore, such a link may exist. An example of such a requirement is illustrated in the implementation section of this paper.

5.3 Development of Individual Modules

Each module to be developed is studied and analysed before a suitable design is produced. The ease of development depends greatly on the design of the framework and the developer tools selected. The next section illustrates the implementation of the

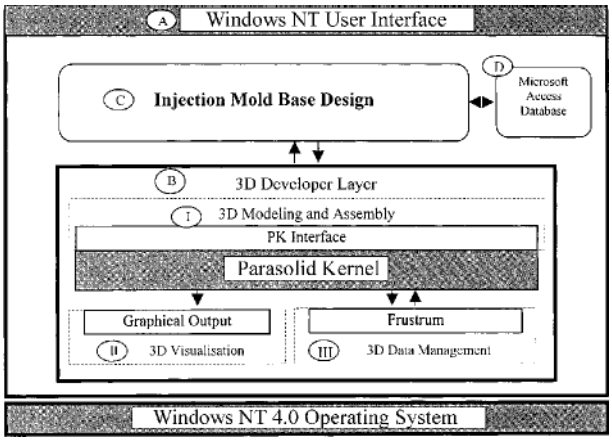

Fig. 3. Overview of the injection mould base design application.

above methodology on a 3D-based injection mould base design and assembly application.

6. Implementations

Applying the system design, a 3D-based injection mould design application is developed. This is achieved using the developmental tools mentioned in the earlier sections. The mould base module is chosen for illustration, as it requires the widest range of 3D functionality, including the generation of assemblies.

6.1 Framework of Application and the Requirements of Each Module

A framework for the application is designed with reference to the developmental work identified. Figure 3 illustrates the framework for the Mold Base design application. The details of the requirements in each module are discussed as follows:

6.1.1 Windows NT User-Interface (A)

Mould base design is an iterative process. The Mould designer first selects a standard mould base from the catalogue, and then repeatedly makes modifications to the dimensions of the mould base until all the design requirements are met. It is, therefore, necessary to consider an interactive user-interface for such purpose. Using the Visual C a n d the MFC, a Windows-based interface is developed. These include:

1. Creation, display and management of menu bar items, context menu items and toolbar buttons for easy access to functionality of the application.
2. Creation, display and management of dialogue boxes to guide the user or to obtain user input.

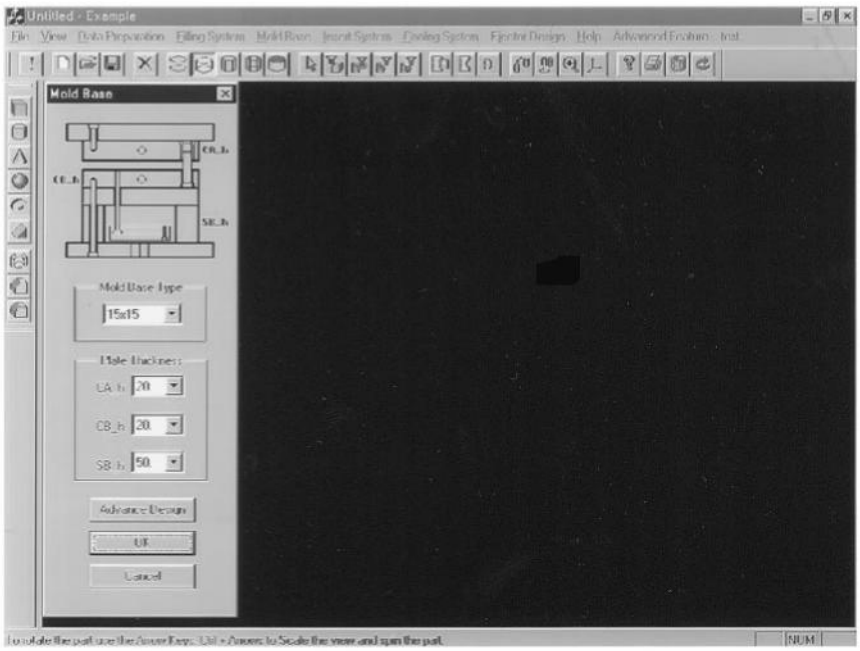

Fig. 4. Windows-based interface.

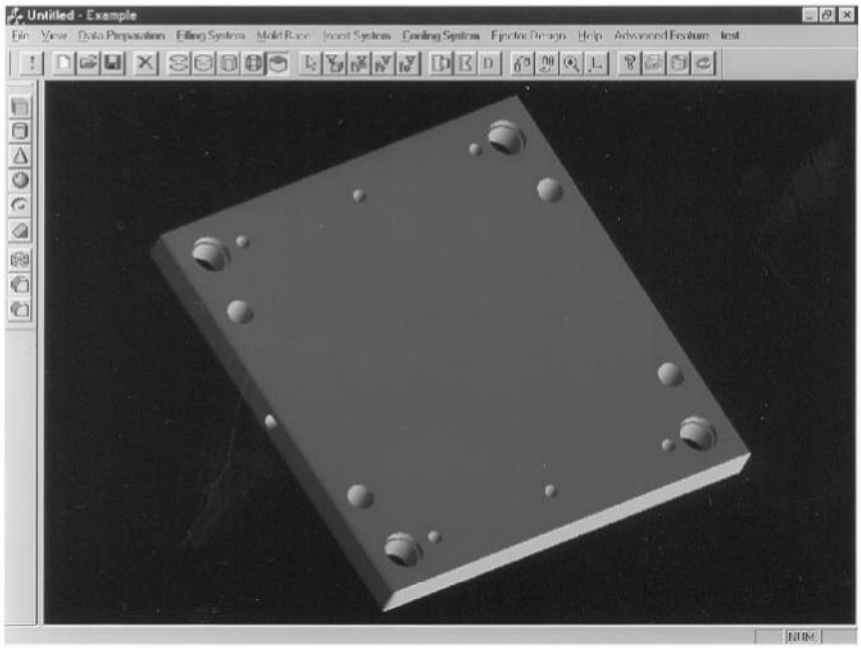

Fig. 6. Cavity plate B.

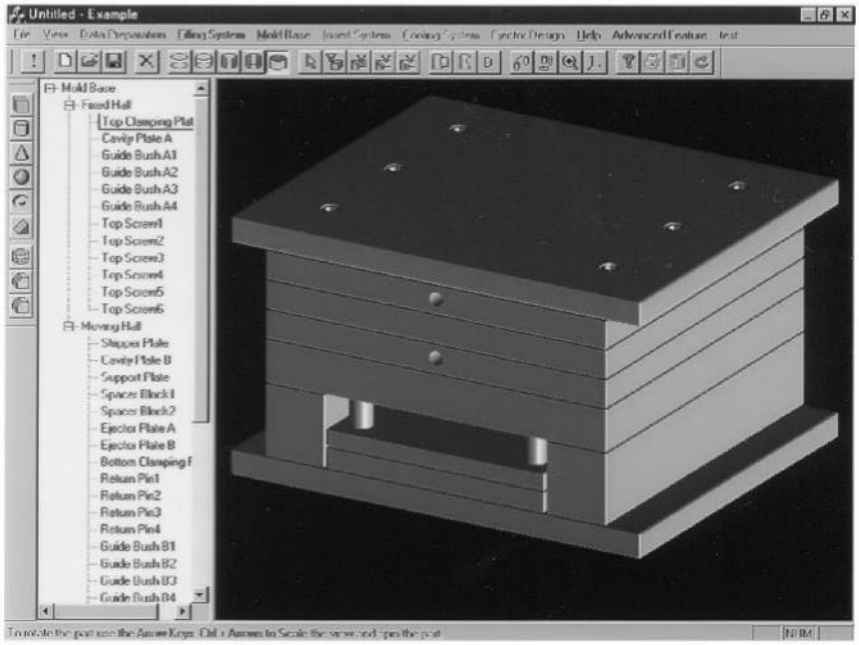

Fig. 7. A"Hoppt" two-plate mould base

3. Creation, display, and management of various views in the display area, for illustration.
4. Mouse driven interaction.
5. The design of the sequence of operation (including user interaction) for each function.

The resulting application, as shown in Fig. 4, is a typical Windows-based application with a user-friendly interface.

6.1.2 3D Developer Layer (B)

The 3D-based requirements of mould base design is analysed and the modules to be developed are identified. The modelling requirements for 3D-based mould base design are:

1. Creation of primitives such as blocks, cylinders, cones, prisms, and toruses.
2. Creation of blends and chamfers.

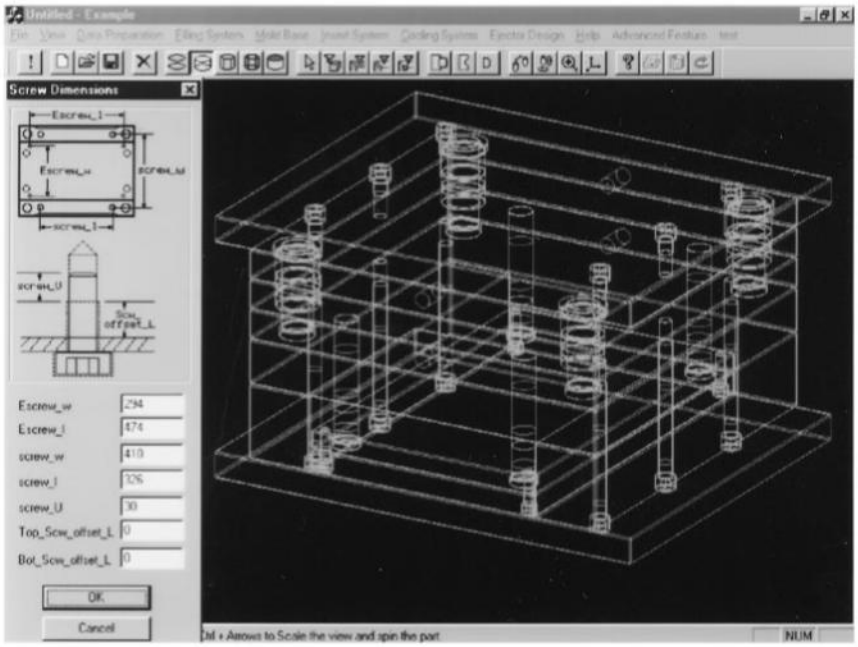

Fig. 8. Customisation of bottom screw dimensions.

3. Boolean operations: unite and subtract.
4. Transformation operations: translation and rotation.
5. Management of object attributes such as name and colour.
6. Creation of instances.
7. Creation of assemblies and subassemblies.

As these are not too extensive, it is possible to develop a basic modelling set. With the detailed development of the individual modules, more functions are then added to the 3D developer layer. The overall requirements in each module are illustrated in the following sections.

I. 3D Modelling and Assembly. A mould base is essentially an assembly of many components such as plates, bushes, pins, and screws. To facilitate mould base design, the designer must be provided with a library of ready-made mould base components. By selecting a particular dimension, a standard mould base will be generated. To facilitate these, a library of 3D-based functions, corresponding to the requirements mentioned in Section 6.1.2, are identified and developed. As the codes are object-oriented, they can easily be expanded to accommodate other mould design modules when required.

II. 3D Visualisation. Using the functions provided in the graphical output, together with OpenGL as the graphical interface, several functions are developed for 3D rendering, view projections and view transformations. These include:

1. Rendering 3D parts with selected colours (Fig. 6).
2. Rendering 3D assemblies with selected colours (Figs 7 and 8 for rendering in shaded and wireframe modes, respectively).
3. Rendering other 3D entities on screen with selected colours.
4. Rendering individual components with a different colour in a mould base assembly.
5. Interactive view transformation such as rotation, translation, and zoom.
6. Assembly tree display and manipulation.

III. 3D Data Management. Portability is one of the benefits of developing a stand-alone application. It is thus important to adopt an open format for maximum portability. The native Parasolid file format (.xmtFtxt) is thus used instead of a new file format. Data management requirements of a mould base module include the following:

1. Open, Save, Save As and Close Parasolid part files.
2. Open, Save, Save As and Close Parasolid assembly files.
3. Import and Export part files.

6.1.3 Mould Base Modules (C)

In order to facilitate the automatic generation of standard mould base assemblies, the application must provide a library of mould base components, whose dimensions depend on standard values found in catalogues. To facilitate design, subsequent modifications to these dimensions have been enabled. The details of this module will be discussed in Section 6.2.

6.1.4 Database Support (D)

A standard mould base requires almost a hundred parameters to completely represent the dimensions and positions of the individual components. Many of these parameters are interrelated and can be derived from others. A database file is thus required to store the catalogue-based parameters of standard mould bases. Microsoft Access database format is used, as there are facilities in the MFC for direct access to Access Database files. Using the Data Access Objects (DAO) in the MFC, a set of functions is developed for the extraction and management of these relevant parameters from the database.

```
void CmoldBaseDesigner::GenrateMoldBase()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                   Individual Components parts are generated first
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
PK_PART_t TopClampingPlate,CavityPlateA,StripperPlate,CavityPlateB,SupportPlate,SpacerBlock,BottomClampingPlate,
PK_PART_t EjectorPlateA,EjectorPlateB,GuideBushA,GuideBushB,GuidePin,ReturnPin,EjectorScrew,TopScrew,BottomScrew;

TopClampingPlate    = m_HopptMoldBase.MakeTopClampingPlate(TCP_h, M_l, M_w, O_W, screw_l, screw_w,
                      top_scw_DS, top_scw_K,top_scw_DK,screw_num);
CavityPlateA        = m_HopptMoldBase.MakeCavityPlateA(CA_h,M_l,M_w,O_L,O_W,GUI_x,GUI_y,GUI_2_offset_x,
                      GUI_2_offset_y,GUI_3_offset_x,GUI_3_offset_y,GUI_4_offset_x,GUI_4_offset_y,Gbush_d1,
                      Gbush_H,Gbush_D,screw_U,screw_w,screw_l,top_scw_ds,eyebolt_d,eyebolt_k,screw_num);
StripperPlate       = m_HopptMoldBase.MakeStripperPlate(B1_h,M_l,M_w,O_L,O_W,GUI_x,GUI_y,GUI_2_offset_x,
                      GUI_2_offset_y,GUI_3_offset_x,GUI_3_offset_y,GUI_4_offset_x,GUI_4_offset_y,Gbush_d1);
CavityPlateB        = m_HopptMoldBase.MakeCavityPlateB(CB_h,M_l,M_w,O_L,O_W,RET_N,RET_x,RET_y,
                      GUI_D,GUI_H,GUI_x,GUI_y,GUI_d,GUI_2_offset_x,GUI_2_offset_y,GUI_3_offset_x,GUI_3_offset_y,
                      GUI_4_offset_x,GUI_4_offset_y,screw_U,screw_w,screw_l,top_scw_ds,eyebolt_d,eyebolt_k,screw_num);
SupportPlate        = m_HopptMoldBase.MakeSupportPlate(SP_h,M_l,M_w,O_L,O_W,RET_x,RET_y,RET_N,
                      top_scw_DS,screw_w,screw_l,screw_num);
SpacerBlock         = m_HopptMoldBase.MakeSpacerBlock(SB_h,SB_w,M_l,M_w,O_L,top_scw_DS,screw_l,screw_w,
                      screw_num);
EjectorPlateA       = m_HopptMoldBase.MakeEjectorPlateA(EA_h,M_l,O_L,EB_w,Escrew_ds,Escrew_w,Escrew_l,RET_O,
                      RET_H,RET_x,RET_y,RET_N);
EjectorPlateB       = m_HopptMoldBase.MakeEjectorPlateB(EB_h,M_l,O_L,EB_w,Escrew_DK,Escrew_K,Escrew_w,Escrew_l,
                      Escrew_DS);
BottomClampingPlate= m_HopptMoldBase.MakeBottomClampingPlate(BCP_h,M_l,M_w,O_W,top_scw_DK,top_scw_K,
                      screw_w,screw_l,top_scw_DS,screw_num);
GuideBushA          = m_HopptMoldBase.MakeGuideBushA(Gbush_d,Gbush_d1,Gbush_D,Gbush_H,CA_h,A01_h,A02_h);
GuideBushB          = m_HopptMoldBase.MakeGuideBushB(Gbush_d,Gbush_d1,Gbush_M,B1_h);
GuidePin            = m_HopptMoldBase.MakeGuidePin(GUI_d,GUI_D,GUI_H,reverse_flag,CA_h,CB_h,A01_h,A02_h,
                      B1_h,B2_h);
ReturnPin           = m_HopptMoldBase.MakeReturnPin(RET_d,RET_D,RET_H,RET_P,RET_N,RET_offset_length,CB_h,
                      SP_h,SB_h,EB_h);
EjectorScrew        = m_HopptMoldBase.MakeEjectorScrew(Escrew_d,Escrew_k,Escrew_K,Escrew_dk,Escrew_ds,Escrew_s,
                      Escrew_t,Escrew_r,Escrew_offset_L,EA_h,EB_h);
TopScrew            = m_HopptMoldBase.MakeTopScrew(screw_d,top_scw_k,top_scw_K,top_scw_dk,top_scw_ds,top_scw_s,
                      top_scw_t,top_scw_r,screw_U,TCP_h,Top_Scw_offset_L);
BottomScrew         = m_HopptMoldBase.MakeBottomScrew(screw_d,top_scw_k,top_scw_K,top_scw_dk,top_scw_ds,top_scw_s,
                      top_scw_t,top_scw_r,screw_U,TCP_h,Bot_Scw_offset_L,BCP_h,SB_h,SP_h,B01_h,B02_h);


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                          Mold Base Assembly is then generated by creating the instances from the Components
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
PK_ASSEMBLY_t MoldBase;
MoldBase = m_HopptMoldBase.MakeMoldBase(TopClampingPlate,CavityPlateA,StripperPlate,CavityPlateB,SupportPlate,
           SpacerBlock,BottomClampingPlate,ReturnPin,EjectorPlateA,EjectorPlateB,GuideBushA,GuideBushB,GuidePin,
           EjectorScrew,TopScrew,BottomScrew,TCP_h,CA_h,CB_h,SP_h,SB_h,EA_h,EB_h,BCP_h,B01_h,B02_h,B1_h,B2_h,
           A01_h,A02_h,A1_h,A2_h,MP_h,M_w,SB_w,GUI_x,GUI_y,GUI_2_offset_x,GUI_2_offset_y,GUI_3_offset_x,
           GUI_3_offset_y,GUI_4_offset_x,GUI_4_offset_y,RET_x,RET_y,Escrew_w,Escrew_l,Escrew_k,Escrew_K,screw_w,
           screw_l,top_scw_k,top_scw_K,screw_num);
```

Fig. 9. Sample code for the mould base designer.

6.2 Development of the Mould Base Design Module

The mould base module consists of three major sections, namely, the mould base component library generator, the mould base assembly generator, and the mould base selection and customisation module. A fourth section, called the mould base parameters manager, is also developed to provide database support for the application. These are illustrated in Fig. 5. The

details of each section are discussed in the following.

I. *Component Library Generator.* With support from the 3D developer layer, standard components for mould bases are created and stored in the component library. By specifying the appropriate dimensions, these components can be generated and used by the mould base assembly generator when required. Figure 6 illustrates a cavity plate created by the components library generator.

II. *Assembly Generator.* Using the 3D developer layer and the component library generator, standard mould bases are assembled and stored in the assembly library. When supplied with a particular parameter set from the database support, specific standard mould base assembly can be automatically generated. Figure 7 shows a "HOPPT" two-plate mould base created by the assembly generator.

III. *Parameters Manager.* The parameters manager acts as a link between the mould base application module and the database support. When a specific standard mould base is selected, the corresponding parameter set for the mould base assembly is extracted from the database file and sent to the component library generator and the assembly generator. Besides this, the parameters manager also allows the parameters to be modified by the users for design purposes. Figure 8 illustrates the modifications of bottom screw dimensions through the interactive user-interface.

IV. *Mould Base Designer.* The mould base designer serves two main purposes. First, to allow the user to select standard mould bases from the assembly generator. Secondly, to facilitate mould base design, by allowing the mould designer to modify dimensions of the selected mould base. The sample code for the function call to generate the mould base in this module is illustrated in Fig. 9. It was noted that the function uses a large number of variables that represent the parameters of the mould base. These are fed into the component generator for the creation of the various mould base components. The assembly generator then uses the components and the parameter set, for the creation of the mould base assembly. As this is outside the 3D developer layer, no direct Parasolid function calls are seen in the sample program.

The current mould base design application is capable of realising all the functionality of the injection mould base designrequirement in a mould design shop. As the mould base is the most 3D-intensive of the IMOLD modules, its successful development implies the feasibility of developing a complete 3D-based injection mould design and assembly application.

7. Conclusion

Advancement in high-level programming languages has allowed programmers to re-use programming codes that are embodied in objects such as the Microsoft Foundation Classes. These powerful features have freed the programmer from the more mundane routines of programming standard functions and creating user-interfaces. They are now able to focus on the core components of the software, thus increasing productivity. This led to the increasing feasibility of developing stand-alone versions of add-in software such as those for CAE, CAD and CAM. Currently, however, such an approach is both timeconsuming and technically demanding. It is nevertheless, feasible and very promising. By integrating the capabilities of several advanced developer tools, we have managed to increase the power of these tools to develop successfully a stand-alone application for injection mould design. So far, only the first three stages of the mould design process have been coded. These form the foundation for the development of the subsequent mould design modules. The methodology applied can also be easily implemented on other software that involves designing with standard components. These include jigs and fixtures design, die casting, and manufacturing line-automation.

References

1. C. K. Mok and Edmund H. M. Cheung, "Computer aided injection mold design using knowledge base approach", Conference Paper, Department of Manufacturing Engineering, City Polytechnic of Hong Kong, 1994.
2. Jami J. Shad, Hiren Dedhia, Viren Pherwani and Sachin Solkhan, "Dynamic interfacing of applications to geometric modeling services via modeler neutral protocol", Computer-Aided Design, 29(12), pp. 811–824, 1997.
3. "The parasolid documentation set", Version 10.1.123, Unigraphics Solutions Inc, 1999.
4. "IMOLD training manual", Version 2.0, Manusoft Plastic Pte. Ltd, 1998.
5. K. S. Lee, J. Y. H. Fuh, A. B. T. Koo and Z. Wang, "A knowledge-based engineering system for the design and assembly of plastic injection mould", Proceedings of 1st National CAD/CAM Conference, KL, Malaysia, 1995.