Advances in Engineering Research, volume 166
3rd International Conference on Automation, Mechanical Control and Computational Engineering (AMCCE 2018)

Construction of Data Parsing System Based on DGN File

Weiya Zhao [1, a)], Jinghua Liu[2, b)] and Shicai Li [3, c)]

[1] *Beihang University Computer Graphics, Beijing 10019, China.*
[2] *Beihang University Industrial Design, Beijing 100191, China.*
[3] *Beijing Zhongke Fulong Computer Technology Co.,Ltd, Beijing 100085, China.*

[a)] zhaoweiya1395@126.com
[b)] ljh1012@126.com
[c)] 18512293@qq.com

Abstract. A efficient data sharing system can not only accelerate the process of software development, the software itself will also be able to compensate for possible vulnerabilities which can greatly improve the quality of software. This paper analyses the storage characteristics and encoding principle of ISFF file format, using the dgnlib open source library to construct a data exchange system, realized data extraction and mapping for the DGN geometric 、DRV property data correctly, final implementated model redraw and storage by our custom file format in actual project.

Keywords: DGN File, storage characteristics, file format, data extraction

INTRODUCTION

As a branch of CAD, the 3D factory collaborative design has been rising rapidly in recent years[1], the continuous improvement of computer graphics technology also make the quality of the model and the rendering effect more outstanding[2]. Microstation V7 use the Intergraph standard file format (ISFF), the basic data file format derived from it is DGN file, customers and the third party developers can create custom programs and read write ISFF files without specific licenses. As we know, PDS as a traditional 3D factory design software[3, 4] has a wide range of applications in the field of design and management of intelligent factories. The basic file formats that it exports are DGN and DRV, former stores the geometric data information and the latter stores the engineering attribute information of the model. Not only focus on extract the geometric data analytic for design file in this paper but also on the key problem about how to achieve the correctly matching for model with attribute after redraw completed in this system.

Our research project is based on the latest development 3D collaborative plant design software platform of the Zhong Ke Fu Long. According to the primitive format defined by both sides and the classification decision of equipment also the differences of the engineering properties in the factory[5, 6] finally realized graphic redraw. After the geometric data and project attribute data imported and mapped correctly, we need to realized serialized storage according to the graph element inheritance relation and model database design[7, 8] of the platform to ensure open model again correctly deserialized.

GEOMETRY FILE STRUCTURE

Element Mode

Referring to the description of ISFF file format, the data structure inside the DGN file[9] is organized as three parts (see FIGURE. 1) macroscopically, including file header、several Element blocks and file end symbol EOD (End Of Design file). Be careful, the TYPE keyword is the key for us to distinguish them.

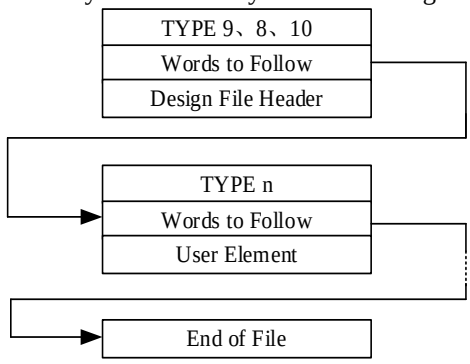


FIGURE 1. DGN file structure

The element block is the direct object we should operate in our follow-up program design, each element block has two most essential attribute types: the type of the element itself and the type of the data structure inside the element.

According to the type of the element themselves, they can be divided into two categories: simple elements and complex elements. There are seven kinds of simple elements: line、 line string、 surface、arc、 ellipse、text and columns; types of complex elements is a collection of simple elements logically. It is composed of two parts, header files and the simple elements which is followed, elements in the header file contains the information of the simple elements, we can use this to determine the size and type of complex elements. In dgnlib, they are given in the way by macro definition.

```
#define DGNT_ARC
#define DGNT_CONR
⋮
#define DGNT_3DSURFAE_HEADER
#define DGNT_BSPLINE_POLE
⋮
```

Structure type within elements is essencely the internal data organize mode, one structure type may correspond to many different elements, so the total structure type is less than the element number, when use dgnlib as a development tool, firstly we shoud judge the structure type which would make the subsequent conversion more reasonable and efficient. FIGURE. 2 represent simple elements, as we can see that no matter line set、surface or Bezier curve, their internal data structure are all MULTIPOINT. This several-for-one structural feature is an important reference for us to design program.

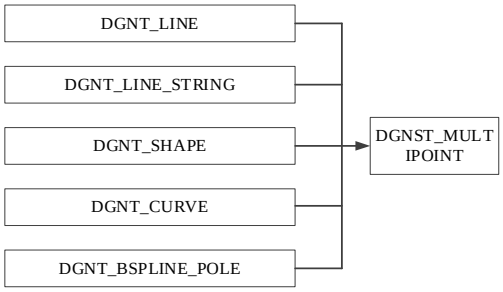


FIGURE 2 . Simple Element Structure Type

Special Byte Characteristics

When reading data in design file, not all data can be directly acquired. Special data need to be reorganized at byte storage level, so we need to figure out the storage mode of its data.

In DGN file, the storage of long type is neither big-endian nor little-endian but a kind of special storage middle-endian shown in FIGURE.3 which is a special features of DGN file, understand this is important for us to crucial restructuring attribute bytes and special quaternion.

2	3	0	1

FIGURE 3. Middle-endian byte ordering

Direction of Graphic Element

In design file, we need to determine the direction through a series of transformations, each primitive graphic has a unique correspond transformation matrix which is stored in a compressed format called quaternion, the quaternion corresponding to a 3x3 orthogonal matrix .Transform according to the following formula:

$$\begin{cases} q[0]=1.0*psElement->quat[1]/(1<<31); \\ q[1]=1.0*psElement->quat[2]/(1<<31); \\ q[2]=1.0*psElement->quat[3]/(1<<31); \\ q[3]=1.0*psElement->quat[0]/(1<<31); \end{cases} \tag{1}$$

In some special cases, the structure of the element does not store quaternion directly. Here, we need to translate the four characters into an integer by pointer location. The transformation macro function code is defined within the program according to the data storage characteristics in the last section:

```
#define DGN_INT32(p) (p[2])\
                     + p[3]*256\
                     + p[1]*65536*256
                     + p[0]*65536
```

Quaternion conversion formula:

$$quat[i] = DGN\_INT32(psDGN-> abyElem + 46 * i) \tag{2}$$

Through the formula we can get the usual sense of quaternion which uniquely determine the rotation matrix of the element, the matrix conversion formula is as follow:

$$\begin{cases} mat[0]=1-2*(q[1]*q[1]+q[2]*q[2]); \\ mat[1]=2*(q[0]*q[1]-q[2]*q[3]); \\ mat[2]=2*(q[0]*q[2]+q[1]*q[3]); \\ mat[3]=2*(q[0]*q[1]+q[2]*q[3]); \\ mat[4]=1-2*(q[0]*q[0]+q[2]*q[2]); \\ mat[5]=2*(q[1]*q[2]-q[0]*q[3]); \\ mat[6]=2*(q[0]*q[2]-q[1]*q[3]); \\ mat[7]=2*(q[1]*q[2]+q[0]*q[3]); \\ mat[8]=1-2*(q[0]*q[0]+q[1]*q[1]); \end{cases} \tag{3}$$

According to the rotation matrix we can get the normal vector、start-stop vector or any direction of the element and finally make sure the element position. For this system, taking Arc as an example shown in FIGURE 4:

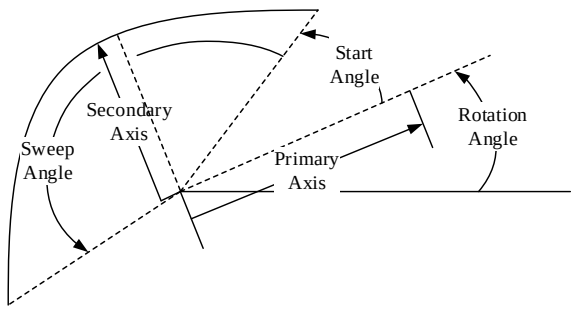


FIGURE 4. Arc parameter

We defined refVector as the result of Primary Axis rotated by Start Angle, and the calculation formula is：

$$\text{refVector}[i] = \cos(\pi * (\text{elemArc->startang}/180)) * \text{mat}[i]+\sin(\pi*(\text{elemArc->startang}/180))*\text{mat}[i+3]; \quad (4)$$

The use of rotation matrix will change based on different actual needs. Reference Arc, we can get any direction vector of the element.

Complex Primitive Judge Mechanism

ISFF as a common file format, we can not clearly determine the specific element type of a complex element just based on a single parameter. Through the dgnlib dump mechanism and the corresponding creat functions, we restore the data of multiple primitives, the results show that any complex primitive has a series of boundary and regular lines which are strictly limited as lines and arcs. They are fixed in the order in the files, starting with two boundary elements followed by a number of regular lines. FIGURE. 5 and FIGURE. 6 respectively show the elemental composition of a rotating entity and an stretch entity. The red lines is the border lines and the yellow is the regular lines:

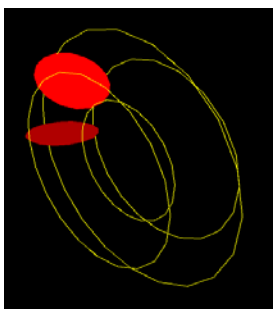


FIGURE 5. Rotation entity

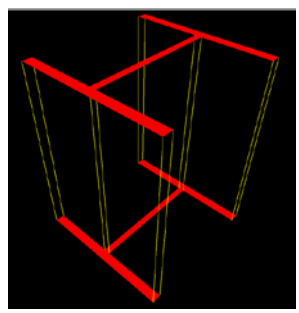


FIGURE 6. Stretch entity

The actual geometry data what we get can are only these simple elements. To determine the specific type of primitives we need define a set of mechanisms, the judge conditions is very important to ensure that the type of primitives correctly. The first is the generation of complex elements just like rotation、projection etc. Currently used surface、solid types are mainly divided by macro as follow:

```
#define DGNSUT_SURFACE_OF_PROJECTION
#define DGNSUT_BOUNDED_PLANE
⋮
#define DGNSOT_VOLUME_OF_REVOLUTION
#define DGNSOT_BOUNDED_VOLUME
```

Then according to the simple element type、number and the arrangement order, if necessary, we need the individual elements geometry data such as arc scan angle, as long as we can judge the conditions as much as possible refinement, the result is almost can not be wrong.

PROGECT PROPERTY

After consulting a large amount of documents and many experiments, the mapping mechanism between each device and the project attributes in DRV file is based on a set of integers which similar to the key-values, whether a device has an engineering attribute is determined by the attr_bytes keyword in the DGN, the progect properties of the device is surrounded by a set of lbl in the DRV file, one of which is shown as follow:

lbl{0 3 640 496 text {Comp no: PST-3
Construction status: New
⋮
Insulation: H(H2)70MMmm
PIPE CLASS:GC2}}

As we can see, the four integers in the first line is the mapping keywords between the properties file and the design file. In DGN file, we use the dgnlib interface to get the attribute keywords pointer as follow:

```
pData = DGNGetLinkage(hDGN,psElement,
        0,NULL,NULL,NULL,&nLinkSize)
```

Next, according to the mid-end storage mode that we described earlier. We do the following conversion:

```
int value[4];
value[i] = pData[2*i] + (pData[2*i+1]<<8);
```

After the transformation, the four integers will match the DRV to achieve the correctly mapping of the attribute data.We use conventional text-based way reading the attribute data in DRV line by line. In our system, use msgpack to realized package storage. According to this reading mode until to the end of one lbl.Concrete examples of this section realized results is shown in FIGURE 7:

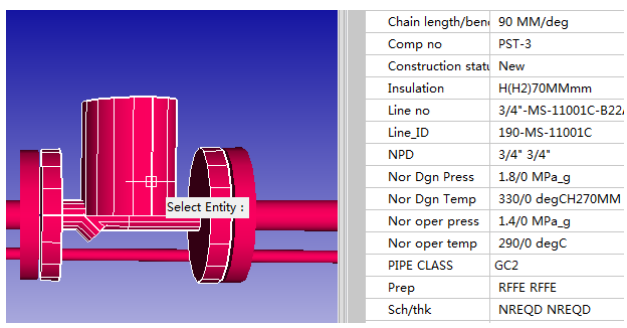


FIGURE 7. Attribute match result

SYSTEM PROGRAM IMPLEMENTATION

System Working Process

When we load the model file, we can load the DRV file meanwhile, for every model element, firstly, figure out whether there is attribute information and searched for in the DRV file, if succeed, judge the type of the structure and the type of the element, then using the judgment mechanism defined by the user self to determine the specific primitive type finally, what should be noted is that the system must consider the possibility of errors in order to have an accurate statistics for the conversion results and data loss when the conversion completed. After all the data preparation is completed, the primitives can be created and redraw. FIGURE.8 is the system work process:

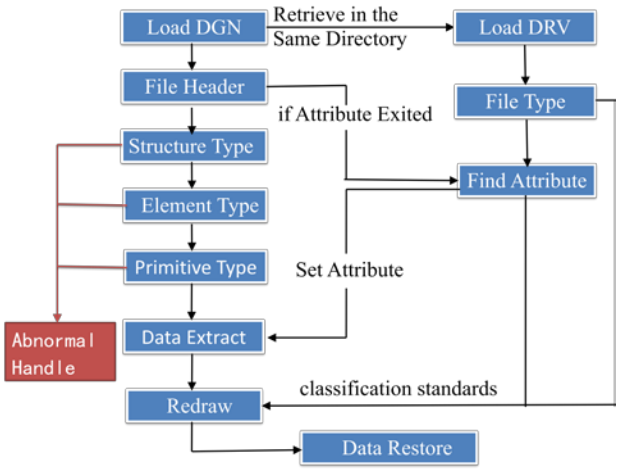


FIGURE 8. System work process

Main Framework of Program

The design of the program relies on the dgnlib open source library interface features and the definition process of our own project graphics platform.Part of the code is as follow:

```
 hDGN = DGNOpen(filePath, FALSE);
DGNElemCore *psElement = NULL;
while((psElement=DGNReadElement(hDGN))!= NULL)
  {
   switch(psElement->stype)
   /* first level standard classification */
       {
         case DGNST_MULTIPOINT:{……}
          /*internal processing functions*/
            break;
         case DGNST_COMPLEX_HEADER:{……}
            break;
                ⋮
        }
    }
```

Here we use the ComplexHeader as an example for specific instructions to explain the second level of classification within the function.

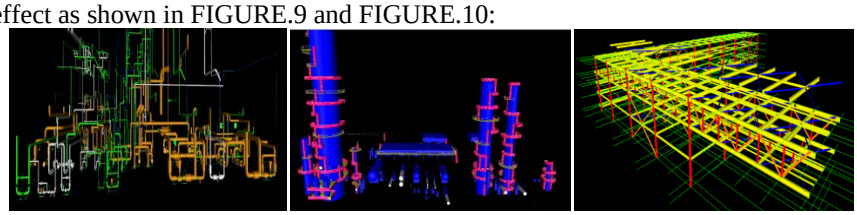
```
    switch(complexHeader->core.type)
    {
      case DGNT_3DSOLID_HEADER:
       {
       /*custom function to judge the complex Primitive*/
       entityType = getEntityType(complexHeader,elemVector);
       /*custom Primitive type by macro*/
       if(entityType == ENTITY_TYPE_SPOLYGON)
       {/*Internal processing functions*/}
       else if (entityType == ENTYTY_TYPE_OVAL){}
         ⋮
       else
       {/*necessary log for data loss scenarios*/}
      for (auto psElement : elemVector)
      DGNFreeElement(hDGN, psElement);
      break;
```

```
        ⋮
    }
```

The main framework roughly is divided into two layers as described above, what we need to pay specially attention is that each layer need to do exception handling, print out the not support information in order to prevent the program collapse.

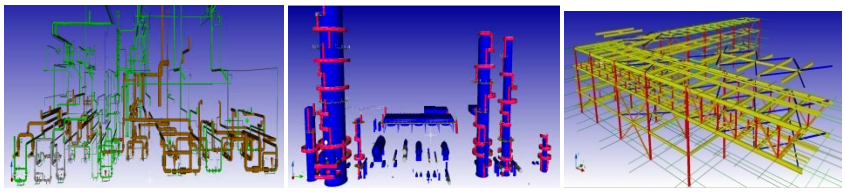
System Working Result

System repaint effect as shown in FIGURE.9 and FIGURE.10:



Piping Equipment Structral

FIGURE 9. Microstation original



Piping Equipment Structral

FIGURE 10. System redraw result

We put the piping、equipment、strutral file which is exported by PDS into the same folder, so that the property files can be automatically retrieved and loaded, the three major types of model files are well supported by the system data parsing and redraw operation, through the actual operation of the program we found that the data extract is not a time-consuming operation, the time cost is mainly spent on the process of redrawing the model. Different draw mechanism of platform may be relatively influence the performance of the system work, therefore, a higher efficiency graphics platform will make the system workig result greatly improved.

CONCLUSION

Through the conversion system designed above, we have greatly realized the DGN model data extraction and model repaint storage, also solved the project attribute mapping、primitive definition etc problems, not only well achieved the purpose of data exchange and sharing but also practical applied to the project. However, the types of primitives supported by the system do not completely cover all the existing models, faced on the variety model files, it is on the cards that the data extraction would occur data loss which is caused by the current system design principles as well as dgn general file format features, and it can not be avoided yet, follow-up we need to further improve the graph library and do a good job for statistics and records of data loss to keep the cost to a minimum level.

REFERENCES

1. Xu, J. H.: Mechanical CAD Technology and CAD Basic Teaching [J]. Journal of Jiaxing College, 2000（6）: 42- 44.
2. Foley J D. Introduction of Computer Graphics [M].DongShihai, Beijing: Mechanical Industry Press , 2005.
3. Liu, Q., Liu, Y.: Talking about the Application of 3D Factory Design System. Information Engineering of Civil Engineering. Vol.4 2012.3.
4. Dai, J.: Research and Application of Parametric CAD System. Shanghai Jiaotong University master's degree thesis,2002,1

5. Xu, Y.X: The geometric basis of computer aided geometric design and manufacture. Jiangsu Science and Technology Press, 1989
6. Zhen, D.G.: Study on the Matching of Factory Design Model Data with Graph Isomorphism Decision. Journal of Engineering Graphics, 2008, No.3
7. Yang, R.Y., Cai, S.J.: Research on Automatic Generation and Application of 3D Digital Architecture. Journal of Intelligent Systems, Volume 3, Number 1, 2008.2
8. Luo, H.: Theory and Practice of Constraint Engineering Parametric Design System for Engineering Graphics. Huazhong University of Science PhD dissertation. 1995,6
9. Yu, H.: Discussion on DGN File Formats. Journal of Geotechnical Investigation & Surveying. 130000