CADmeister 表示フレームワークの高速化

Improvement of CADmeister Display Framework

高　木　　健

要　約　CADmeister の表示フレームワークは製品形状である 3 次元図形や，寸法や注記などの製図情報である 2 次元図形を，CAD 画面上に表示する基盤ソフトウエアである．図形の表示更新機能は，CAD を使用した設計ではコマンド実行の都度動作する欠くことのできない機能である．このため表示更新のレスポンスが悪いと，操作者が CAD システムの応答性に大きな不満とストレスを抱くことになる．

近年のユーザ設計手法の主流がフルアセンブリ設計となったことで，CADmeister 上でも大規模データを扱う機会が多くなっている．こうしたニーズに応じるべく，大規模データを扱う上での課題となるメモリ使用量の改善と画面更新速度の向上を図り，表示フレームワークの高速化を実現した．

Abstract　Display framework of CADmeister is base software that provides function of display of 3D figure as design shape and 2D figure as drafting information such as dimension and annotation on CAD (Computer Aided Design) screen. Display update function of figure is essential in designing using CAD and it operates at every command execution. Thus if response of display update is slow, the situation produces operators' disappointment and stress about response of CAD system.

Mainstream of user design method of recent years is full-assembly design and there are many occasions to use large-size data in CADmeister. To meet these needs, UEL Corporation has tried improvement of memory usage and screen update speed that become issue when using such large-size data and has realized speeding up of display framework.

1. は　じ　め　に

CADmeister の CAD（Computer Aided Design：コンピュータ支援設計）フレームワークは，主として図 1 のように対話操作機能を司る「モニタモジュール」，データベース入出力操作を司る「データベースモジュール」，画面上への 3 次元図形の描画などを司る「表示モジュール」から構成されている．これらの CAD フレームワークはアプリケーションに必要な共通基盤機能を提供し，CAD システムとして統一された操作性とデータベースを実現している[1]．

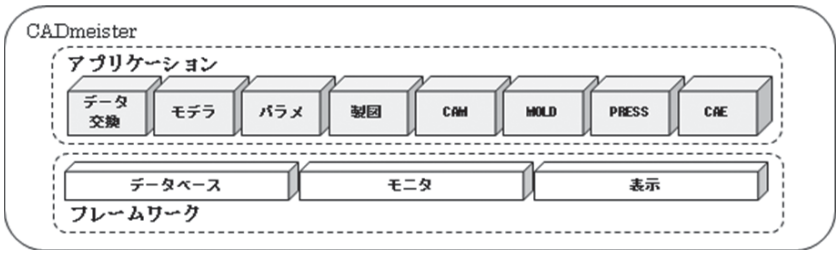

図 1　CADmeister のソフトウエア構成

CAD フレームワークが提供する機能は CAD システム上の全アプリケーションから利用されるため，その処理効率が CAD システム全体の応答性を左右する．とりわけ表示処理はコマンドの実行の都度，画面上の図形表示 Window に表示される形状をリアルタイムに更新するため，高い処理効率が求められる．画面更新の処理レスポンスが遅いと，設計リードタイムが伸びるばかりか，ユーザがコマンド操作にストレスを感じて，CAD システムに不満感を覚えることにもなる．

近年のハードウエアの低価格化と性能向上は著しく，高性能なコンピュータシステムをユーザが導入しやすい環境も整ってきた．加えて設計部品の複雑化・高品位化と設計部品全てを組み上げたフルアセンブリ設計も進んだことで，CAD システム上の設計データサイズは急激に大きくなっている．こうしたデータの大規模化に対応するために，CADmeister の表示フレームワークの高速化を実施した．

本稿では，2 章で現状の課題を分析し，3 章で高速化に向けて実施した四つの施策について述べ，4 章で高速化の成果を発表する．

2. 課題と分析

大規模データでは，表示データ量も膨大であり表示処理には多くのメモリと処理時間を必要とする．従ってこれを扱う CAD システムの表示フレームワークには「大量のメモリが扱えること」と，「高速な画面更新が可能なこと」が要求される．

2.1 表示に要する大量メモリ

CAD システムの表示フレームワークは処理レスポンスを最優先とするため，表示対象形状に関する情報は全てメモリ上に展開・保持するように設計していることが多い．この保持情報には位相関係情報といった形状間の管理情報だけでなく，形状の色や，可視性といった属性，3 次元グラフィックスに必要な構成点情報なども含まれ，多くのメモリを必要とする．

また，CADmeister も含めた他の多くの CAD システムの表示フレームワークは，3 次元図形を描画する標準ライブラリに OpenGL を利用している．OpenGL では「DisplayList 描画」と「Immediate 描画」という二つの描画方法が提供されている（図 2）．

「DisplayList 描画」では，複数の描画命令を予め描画前にグラフィックシステムへ通知してカプセル化する．この時にカプセル化した単位で一意の識別子（以下，DisplayList ID と呼ぶ）を発番するが，その後の描画処理タイミングでは描画対象とする DisplayList ID のみをグラフィックスシステムへ通知して描画を行う．一方の「Immediate 描画」はグラフィックスシステムへの描画命令を描画更新の都度発行し即時描画を実行する方法である．

両描画方法にはそれぞれ次のメリット/デメリットが存在する．「DisplayList 描画」のメリットは，一度カプセル化した内容で再描画を行う際「DisplayList ID」のみグラフィックスシステムに通知すればよいため，フレームワークプログラムからグラフィックスシステムへの命令通知に必要な処理時間が最小で済むことである．デメリットは，カプセル化により集約された描画命令がグラフィックスドライバ（アプリケーションプロセス内）のメモリに蓄積・保持された状態となり，大規模データにおいては大量にメモリを消費することである．

「Immediate 描画」のメリットは，描画命令を画面更新の都度に発行するため，表示内容を描画時点でダイナミックに変更することができることであり，デメリットは，画面更新ごとに

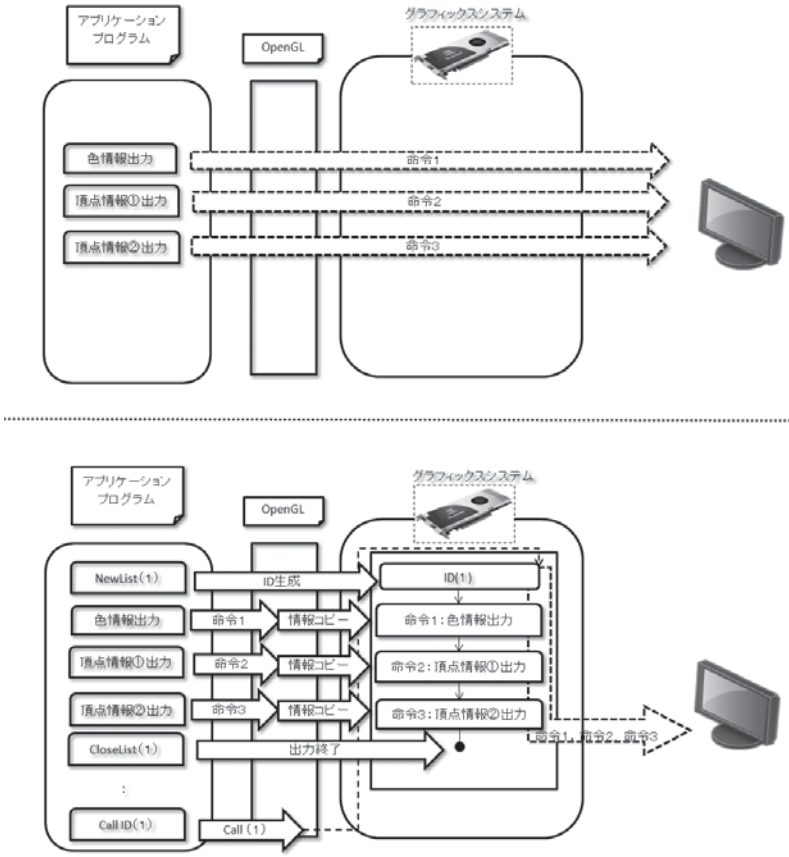

図 2　Immediate 描画（上段）と DisplayList 描画（下段）

全描画対象への描画命令を発行するため，グラフィックスシステムへの描画命令の通知処理時間が大きくなることである．

つまり，それぞれの内容を踏まえると，「DisplayList 描画」は，対象形状そのものの変更頻度が少なく，対象以外を含めた全体描画を頻繁に繰り返し行うケース向きであり，「Immediate 描画」は，対象形状そのもの変更頻度は頻繁だが，対象以外も含めた全体描画の頻度は少ないケース向きであると言える．

CAD システム上では様々な操作で図形が再描画されるが，その中でユーザが最も頻繁に行う操作は設計対象図形を画面上で視認・編集しやすい位置に移動させる回転・平行移動・拡大といったビューイング操作である．この操作ではマウスポインタの移動に追従したリアルタイムな再描画が要求されるが，このとき形状の構成座標点や，色・可視性といった図形そのものの描画内容の変更は必要ない．

つまり，ビューイング操作はカメラ位置を定義するパラメタ（マトリクス）指示命令の変更こそ行うが，形状自体は一度定義した情報を使って再描画すればよい．このような対象形状データそのものに変化のない静的な再描画では前述の通り「DisplayList 描画」が最適な描画方法である．しかし，「DisplayList 描画」は描画時間が高速な半面，描画命令をメモリ上に蓄積・保持する特徴を持つ描画方式でもあるため，大規模データでは多くのメモリが必要になる．表示に要するメモリが大量となることは，時として CAD システムのプロセスメモリが 32bit

アプリケーションのユーザメモリ空間の上限である 2GB を超え，アプリケーションの異常終了をも発生させる．

2. 2　大量データ表示に要する画面更新時間

前節で述べた「DisplayList 描画」では「上位の DisplayList 内の命令群に下位 DisplayList ID の呼び出し命令を含める」ことが可能である．これを利用して表示構造に階層関係を持たせ，再描画時に上位表示構造の DisplayList ID のみ指示して下位の表示構造すべてを描画することができる．たとえば，DisplayList が図 3 の表示構造を成す場合，アプリケーションからは最上位の「A」の DisplayList ID の描画を指示するだけで「B」以下，「G」以下の階層構造すべてが再描画される．DisplayList でこのような表示構造を作成するメリットは，再描画時にアプリケーションからグラフィックスシステムへの通知命令が一回で済むことである．つまり，「Immediate 描画」と比較して再描画に必要な命令数が少なく（= 命令の転送処理時間が小さい），グラフィックスシステムですべての処理が完結することで高速な画面更新が実現できるのである．しかし，このような構造を用いても表示データ量に比例したグラフィックスシステムの命令処理負荷の増大を防ぐことはできず，大規模データでは速度低下が顕著となる．

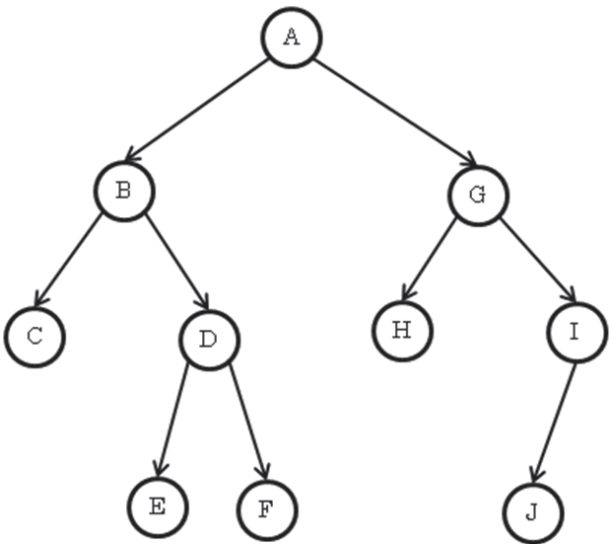

図 3　表示構造

3.　高速化に向けた対策

CADmeister では，表示更新の効率の良さから DisplayList 描画方式を採用していた．しかし，前章で述べたように表示データが大規模化すると，DisplayList の問題点が顕在化した．表示フレームワークの高速化では，更に近年のハードウエア性能を活かすことも考慮し，以下の 4 項目の対策を実施した．これらについて本章の各節で解説する．

① 「DisplayList 描画」から「Immediate 描画」へ描画方式の転換
② 「Immediate 描画」において課題となる「描画命令数の削減」
③ 並列処理（マルチスレッド）化
④ Native 64bit 化

3. 1 「Immediate 描画」への転換

「DisplayList 描画」から「Immediate 描画」へ描画方式を転換するために，表示フレームワーク内の情報管理テーブルの再設計を実施した．「DisplayList 描画」では，DisplayList としてグラフィックスシステムのメモリに保持する一部の情報は，重複保持を避けるためにフレームワーク内メモリには保持していない．しかし「Immediate 描画」では画面更新毎に全描画命令の再発行が可能であることを要求されるため，これまで保持対象外だった情報も新たにフレームワーク内メモリに保持する必要があった．また，画面更新時には，表示構造のトップから最下位まで，順次検索して描画する機能も必要なため，全描画情報の「保持」と「検索」が可能になるように設計した．

3. 2　描画命令数削減

前章で述べたように，「Immediate 描画」では画面更新時に発行される描画命令数の総量に比例して画面更新時間が決定されるため，これを効率よく実行するためには描画命令数を削減する必要がある．この対策として，Culling（間引き）処理や一括描画処理を採用した．

1）　View Frustum Culling

OpenGL で描画する 3 次元図形は，図 4 で示すビューボリュームと呼ばれる範囲（カメラ視界内）のみ画面上に表示される．ビューイング操作による回転や拡大を実施すると，完全にビューボリュームの外に位置する図形も存在する．このような図形は表示する（描画命令を発行する）必要がないため，ビューボリューム外にあるかどうかを判定し，図 5 のように外側と判定された場合に描画命令の発行をスキップするようにした[2][3]．

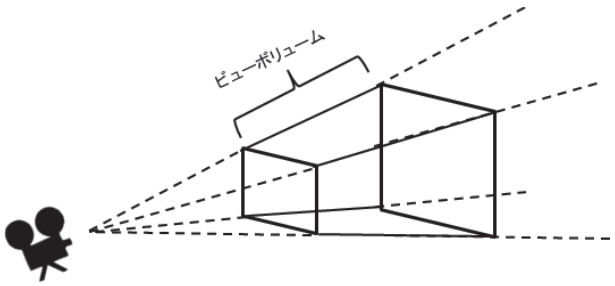

図 4　ビューボリューム

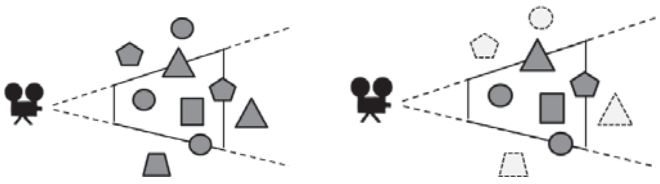

図 5　View Frustum Culling

2） 微小要素カリング

画面上で視認することが困難なほど小さく表示される図形は非表示としても利用ユーザにとって支障はない．たとえば図６の四隅にあるボスは，縮小表示した際には見えなくても支障はない．従って，ビューイング操作の後，３次元図形が画面上一定サイズ以下になったかどうかを判定する処理を追加し，微小要素には描画命令を発行しないようにした．

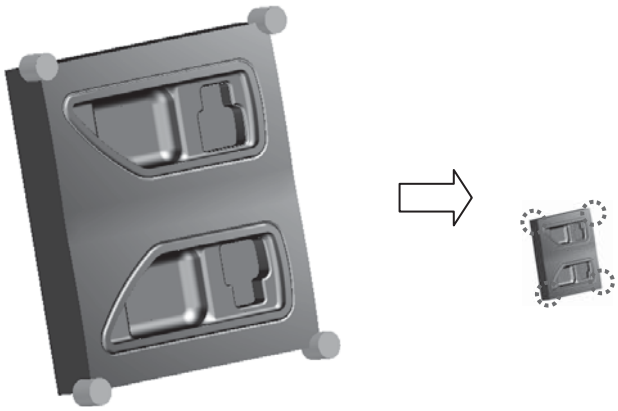

図６　微小要素カリング

3） シェーディング表示中のビューイング操作時の面シンボル/面境界線の描画抑止

面/立体といった形状は，回転・移動・拡大/縮小などのビューイング操作の実行中は面/立体の位置が把握できさえすればよい．このため図7のように，ビューイング操作中はシェーディング表示されている面/立体の面シンボル/境界線を表示しないようにした．

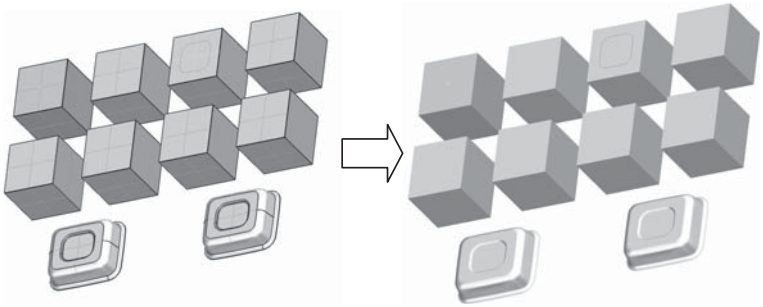

図７　面シンボル/境界線を非表示

4） 縮小表示時の寸法・注記テキストの簡易描画

図面データを縮小表示した場合，テキストを表示しても見づらく，テキストの存在と位置の把握ができれば十分と考える．テキスト図形は各文字がストロークフォントと呼ばれる複数の線プリミティブで構成されており，大量表示されている場合にはグラフィックスシステムに対する負荷も大きい．そのため図面データの縮小表示では，図８で示すように寸法文字や注記のテキスト部分は領域を囲む矩形図形で表示するようにした．

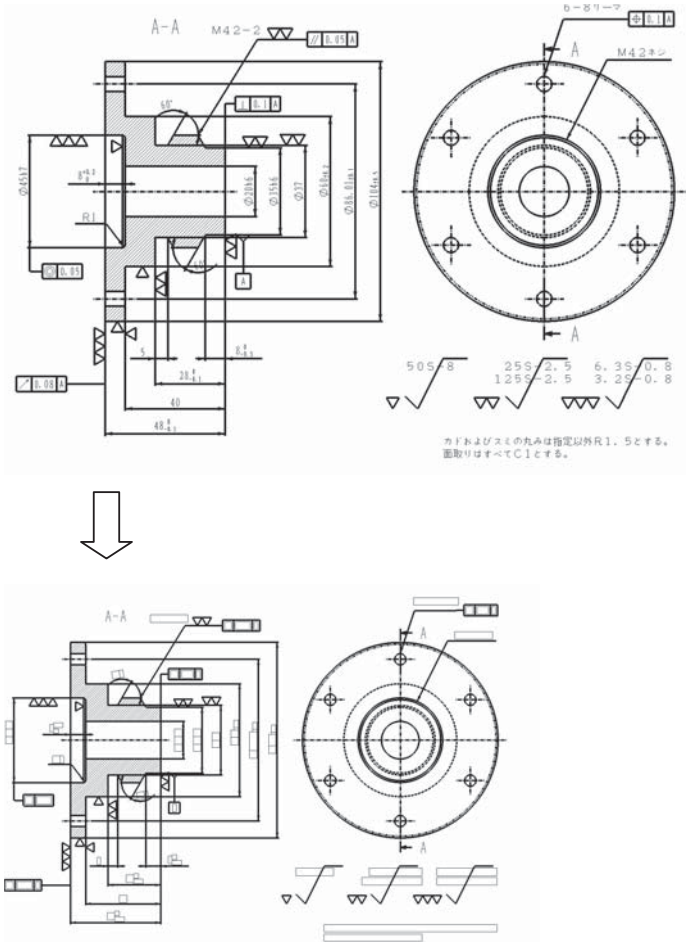

図 8　テキストの簡易表示（上：通常表示，下：簡易表示）

5)　属性別の描画処理の追加

描画対象の色や線種・線幅などの属性が直前の描画対象と異なる場合には，OpenGL に対して属性の変更命令を発行する必要がある．属性の変更命令の発行回数が増えると画面更新の時間が増えるため，予め同一属性かつ同一表示データ種別ごとにソートする最適化処理を追加して，描画命令の発行回数を削減した．

6)　グラフィックスメモリ利用（VBO 利用）

「Immediate 描画」では，描画毎に発生するグラフィックスシステムへの頂点データ転送処理時間を小さくすることが画面更新時間短縮に有効である．描画命令数の削減は 1)～5) の対策を実施したが，更に近年の OpenGL ライブラリでサポートされている VBO（Vertex Buffer Object）を活用して時間短縮を図った．

VBO はグラフィックスシステム上に搭載されている VRAM メモリを頂点座標格納用のメモリとして利用する機能である．あらかじめグラフィックスシステム上のメモリをリザーブして頂点データを格納しておくことで，アプリケーションプロセス内のメモリ使用量の削

減が可能となるばかりか，描画時に必要な対象図形を構成する頂点情報も，CPU-GPU 間の低速な経路（バス）を跨いでではなく，グラフィックスシステム内の高速なメモリ間で転送が可能となる（図 9）．この VBO に対応した表示フレームワークとしたことで，画面更新時間の大幅短縮を実現した．

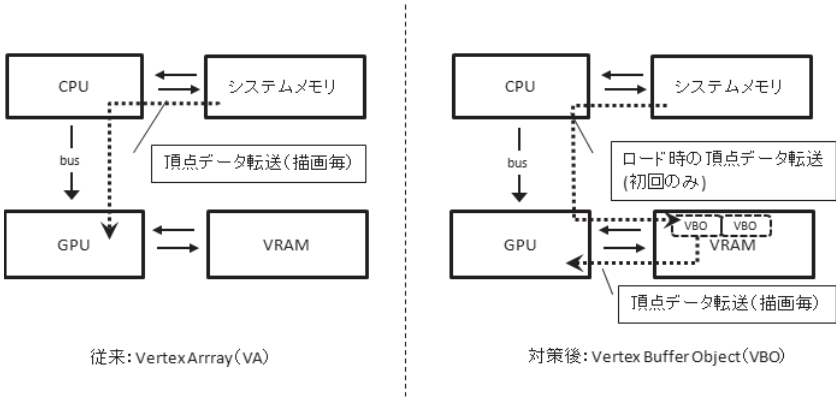

図 9　Vertex Array（VA）と Vertex Buffer Object（VBO）

3. 3　並列処理（マルチスレッド）化

コンピュータシステムの演算性能向上の手法として，CPU の高クロック化は消費電力や発熱量の増大から限界を迎えており，近年は複数の CPU にタスクを分散処理させて総合的な性能向上を狙うマルチコア（マルチプロセッサ）化が進んでいる．マルチコアのコンピュータシステムでは，アプリケーションも複数の CPU を効率よく利用するための並列コンピューティング対応が欠かせない．

CADmeister の多面体近似（シェーディング）処理は，従来，構成面単位に順次計算していたが，最大八つの CPU を駆使して並列処理（マルチスレッド処理）をするように改修して処理時間を短縮した（図 10）．

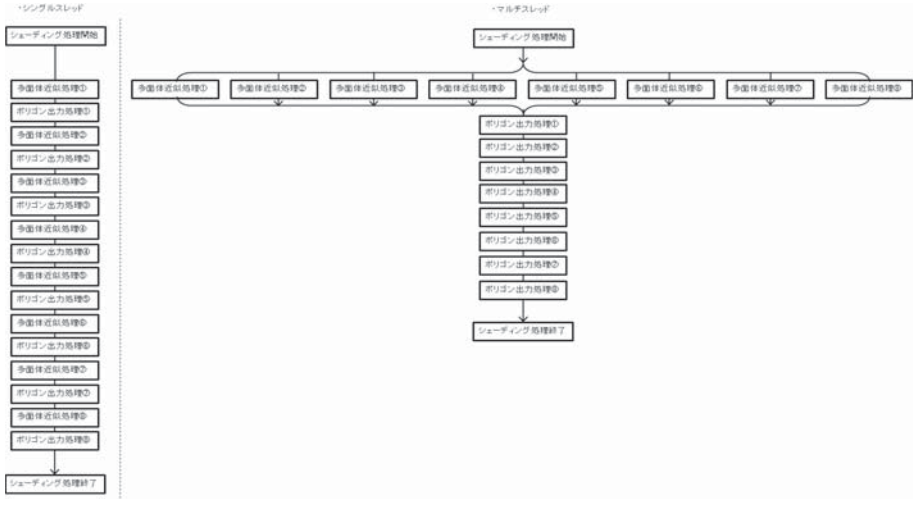

図 10　シェーディング処理の並列化

ただし，OpenGL にはスレッド間を跨って，自スレッドから別スレッドに対し描画命令を発行することができない制約があるため，図 10 に示すように生成されたシェーディング（ポリゴン）用頂点データの VBO 転送などは，全ての構成面の多面体近似処理が終了した後に一括して発行している．

3. 4　Native 64bit 化

32bit アプリケーションとして構築されたソフトウェアは，扱えるメモリ空間が 4GB（実際にアプリケーションが利用できるメモリ空間は通常 2GB）までである．この制限を解消する抜本的な方策として，CADmeister のアーキテクチャを Native 64bit 化した．

一般に 32bit アプリケーションと比べ，64bit アプリケーションでは利用できるメモリ空間が 16TB と飛躍的に大きくなる．この対応によって，64bit OS がサポートする広大なメモリ領域を活用し，アプリケーションとしては理論上 8TB までのメモリを利用することが可能となった．

4.　効果

CADmeister の表示フレームワークに，前章で述べた四つの対策を実施した効果を示す．表 1 は測定したモデル，表 2 は効率測定結果である．高速化前と比較して，メモリ使用量は平均 18.5％の削減，画面更新時間は平均 42.8％の短縮，多面体近似時間は平均 25.5％の短縮を達成し，64bit 版では 32bit 版と比較して 3 倍以上のデータ量を表示できるようになった．

表 1　測定モデル内容と測定機構成

【測定モデル】

測定モデル	データサイズ	部品数	要素数	
モデル 1	430MB	1個	複合図形	4, 343個
			境界線	144, 042本
			構成面	34, 386面
モデル 2	900MB	2853個	複合図形	10, 763個
			境界線	278, 797本
			構成面	120, 419面

【測定機構成】

OS 名:　Microsoft Windows 7 Ultimate
プロセッサ:　Intel64 Family 6 Model 26 Stepping 5 GenuineIntel ~2661 MHz (Xeon W3520)
グラフィック:　NVIDIA Quadro FX 1800
物理メモリの合計: 18,415 MB
HDD:　Seagate ST3500418AS (500GB SATA300 7200)
SSD:　Intel X25-E Extreme SATA Solid-State Drive(リード 250MB/s、ライト 170MB/s)

表２　効率測定結果

【測定結果１】（注1）メモリ使用量（MByte）

測定モデル	表示状態	CADmeister（高速化前）	CADmeister（高速化後）	向上率
モデル１	ワイヤ表示	949	**884**	6%
	シェーディング表示	1152	**912**	21%
モデル２	ワイヤ表示	1726	**1441**	16%
	シェーディング表示	2315	**1489**	36%

【測定結果２】（注1）画面更新時間（ミリ秒）

測定モデル	表示状態	CADmeister（高速化前）	CADmeister（高速化後）	向上率
モデル１	ワイヤ表示	48	**33**	31%
	シェーディング表示	77	**66**	14%
	半透明表示	270	**134**	50%
モデル２	ワイヤ表示	140	**79**	44%
	シェーディング表示	246	**126**	49%
	半透明表示	1021	**316**	69%

【測定結果３】（注1）多面体近似時間（秒）

測定モデル	表示状態	CADmeister（高速化前）	CADmeister（高速化後）	向上率
モデル１	ワイヤ→SHD表示	97.03	**84.51**	13%
モデル２	ワイヤ→SHD表示	48.04	**29.86**	38%

【測定結果４】同時データ表示（シェーディング ON 状態）限界数とメモリ使用量

測定モデル	CADmeister 32bit		CADmeister 64bit	
	表示上限個数	メモリ(MByte)	表示上限個数	メモリ(MByte)
モデル１	3	2998	**16(注2)**	**12240**
モデル２	4	3090	**13(注2)**	**11710**

（注１）【測定結果1】【測定結果2】【測定結果3】は32bit版CADmeisterで測定.

（注２）一部の表示管理IDがシステム上限を超え，エラーが発生したため測定を中断.

5. お　わ　り　に

CADmeister は，今回の改善により大量データも高速に表示できるようになった．しかし，CAD の設計データ量は今後さらに増加すると予想しており，一層の高速化が必要だと考えている．そのための方策として，画面更新時間の短縮では Occlusion Culling[*2] や LOD[*3] の実施を検討している．また，並列プログラミング化の多面体近似以外への適用や，CUDA[*4] など GPGPU[*5] 技術の利用も考えている．さらに効率化を図り，顧客に快適な CAD 操作環境を提供していきたい.

＊ 1　Khronos グループが策定しているグラフィクスハードウェアのアプリケーションプログラミングインタフェース．2 次元・3 次元コンピュータグラフィックス両方が扱える．元々は，Silicon Graphics 社が開発していた．
＊ 2　対象物体が他の物体によって視界からさえぎられるか否かを判定し，隠れる場合は対象物体の描画処理を省略する方法．
＊ 3　LOD（Level Of Detail）とは視点から対象物体までの距離によって描画する際のポリゴン数や，折れ線のベクタ数を増減させること．
＊ 4　CUDA（Compute Unified Device Architecture：クーダ）とは，NVIDIA が提供する GPU 向けの C 言語の統合開発環境であり，コンパイラやライブラリなどから構成されている．
＊ 5　GPGPU（General-purpose computing on graphics processing units；GPU による汎目的計算）とは，GPU の演算資源を画像処理以外の目的に応用する技術のこと．

参考文献

［1］ 中辻 等, 笹尾 忍, 古澤 裕一, 坂東 司, 小竹 正行,「CADCEUS のフレームワーク」, ユニシス技報，日本ユニシス，Vol.14 No.4 通巻 44 号 1995 年 2 月，P98 ～ 119
［2］ “Efficient View Frustum Culling”, Daniel Sýkora and Josef Jelínek, CESCG-2002, http://www.cescg.org/CESCG-2002/DSykoraJJelinek/index.html
［3］ “Open GL Optimizer Programmer’s Guide: An Open API for Large-Model Visualization”, Silicon Graphics International Corp. 1998. 6, http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi/0640/bks/SGI_Developer/books/Optimizer_PG/sgi_html/index.html
（上記参考文献の URL 確認：2012 年 11 月 2 日）

執筆者紹介

高　木　　健（Ken Takagi）
1993 年日本ユニシス・エクセリューションズ(株)入社．翌年より CADCEUS の表示系フレームワーク開発に従事，2004 年から CADmeister の表示系フレームワーク開発に取り組む．