DBMS を用いた仮想 CG 空間の空間情報管理とその評価

高木　良成†　　富井　尚志‡

†横浜国立大学　大学院環境情報学府　情報メディア環境学専攻

‡横浜国立大学　大学院環境情報研究院

E-mail:　†takagi@arislab.dnj.ynu.ac.jp,　‡tommy@ynu.ac.jp

あらまし　本研究では仮想 CG 空間を DB に蓄積して一つの「コミュニティ」を再現することを考える．そのためには 3 次元 CG で作成された世界を格納するための DB スキーマ定義が必要であり，つまりは仮想世界のモデル化が必要となる．本研究でのモデル化手法として，対象世界に存在する意味情報の明示的な定義として ontology を導入し，形状データと意味情報を分離する方法をとる．そこで本研究では，このモデル化手法を用いて仮想世界をモデリングし，実際に DBMS を用いて DB 上に仮想世界を構築する．そして検索・更新に対するシステム内部での操作を規定し，システムとして実装する事で，このモデル化手法の実現性ならびにシステムの有用性を示す．

キーワード　マルチメディアデータベース，空間 DB，コンピュータグラフィックス

Management of Spatial Data in Virtual CG Space with DBMS and Evaluation of Implemented System

Ryosei TAKAGI†　　Takashi TOMII‡

†‡Graduate School of Environment and Information Sciences, Yokohama National University

79-7 Tokiwadai, Hodogaya-ku, Yokohama 240-8501 Japan

E-mail:　†takagi@arislab.dnj.ynu.ac.jp　‡tommy@ynu.ac.jp

Abstract　This paper presents a methodology to produce a cyber “community” by accumulating virtual CG space to database. Therefore, we have to schematize a world created by 3D CG, in other words, we need to design a model of virtual world. We introduce modeling with ontology that defines semantic information of objective community explicitly, and we divide spatial data from semantic information. In this paper, by using this modeling methodology, we actually implemented the virtual world supported by DBMS. And we define functions for inquiry and update in the implemented system, which shows usefulness of proposed methodology of modeling and the system.

Keyword　Multimedia DB, Spatial DB, CG

1. はじめに

近年の高度なグラフィックコンピュータの登場とマルチメディア技術をもとに 3 次元グラフィックスやバーチャルリアリティ分野の研究が進められており[1], 仮想美術館など VR システムの実用化がなされて来ている．これらの研究の多くは視覚性や体感性など，利用者がいかに現実感を得られるかに焦点が当てられている．一方で 3 次元グラフィックスのデータに対し位置や動きなどをキーとした検索を行ったり，現存するデータに対する更新のサポートなどを考えると，データベースとしての高度な利用システムについて考慮する必要がある．3 次元の世界を対象にしたデータベースシステムとして RWDB[2]や VWDB[3]などの研究が行われてきた．そこで本研究では、データベースを用いると共にさらに共有空間という概念を取り入れ CG で作られた仮想空間を使って一つの「コミュニティ」を再現することを考える．

そこで考慮されるべき事項はユーザに視覚的に情報が提供されることと，ユーザの意思疎通が可能となることである。この二つを実現する手段として 1.外部とのデータのやりとり 2.システム内部での操作, 実装 3.概念的な情報構造に基づいたモデル化の 3 層からとらえ DBMS を用いてこれを実現すると考える．

そこで，3.に対し，3 次元仮想空間を構成する VRML などの従来の仮想空間記述方式を考えてみると，意味情報はオブジェクトの付加属性として表現されることが多い．しかしこの手法では，付加情報値が多岐に渡り，また意味情報そのものが体系化されていないため，

(1)構造を持った意味情報を記述することが出来ない．(2)モノが持つ意味が同じであるか異なっているかを付加属性内の文字列マッチングでしか表現できない．(3)モノの数が多くなったとき，付加されている意味情報の全検索になるので計算に時間がかかる可能性がある，など問題点が多い．

そこで本研究では，意味情報を仮想空間の物体の形状情報から分離するモデル化手法を提案する．具体的には，あらかじめ「コミュニティ」空間に必要である意味情報を抽出し，意味情報の関係を明示的に定義する．このような「概念化の明示的な規約」はontology[4][5]とよばれる．このontologyに基づいて仮想世界をとらえた際の，仮想世界に存在する「もの」をmediatorとして作成する．そして実際の仮想空間の物体の形状情報(ポリゴンデータ)をraw dataとする．

本研究では，この3層のモデル化手法の実現性と有用性を示すために，実際にMicrosoft社製SQLserver2000を用いてシステムを実装し，検索更新が可能であることを示す．具体的には本モデル化手法でモデリングされた「コミュニティ」空間をSQLserver2000上に実装し，検索更新に対するRDBMSを用いたシステム内部での操作を定義して，検索結果をCGとして表示する，またユーザのブラウザからの更新をデータベースに反映させる．以上により本モデル化手法およびシステムの有用性を示す．また高速ネットワークを用いて，質問処理に要する一連の処理時間のベンチマークを行い，「コミュニティ」での利用の可能性を検証する．

2. DBMSを用いたコミュニティ空間のアーキテクチャ

まず本節では，1.に述べた「コミュニティ」空間について説明する．

本研究室ではDBMS上に構築された3次元仮想世界を，複数のユーザが共有し，ユーザは仮想世界に対しad-hocな検索や更新を行う事ができるシステムを考える．そのアーキテクチャを図1に示す．

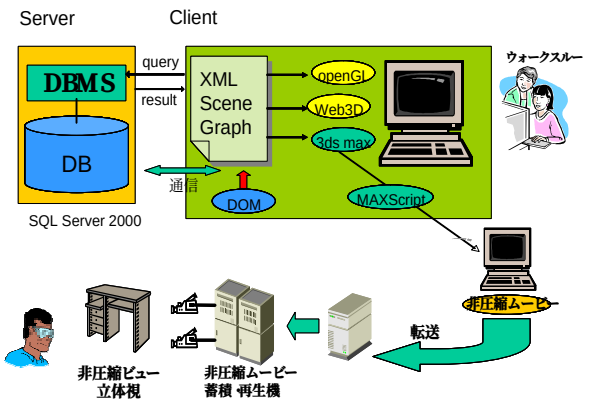

図1. コミュニティ空間のアーキテクチャ

まずDBserverでは概念的な情報構造に基づいてモデル化された仮想空間がSQLserver2000上に蓄積される．次にそのDBserverに対し行われた検索の結果は，複数枚のテーブルとして取り出される．そのテーブル群を，利用者端末に転送された際XMLによって記述された階層構造を持つシーングラフに変換する．このシーングラフは利用者が使用するブラウザ(OpenGLブラウザや3ds max (discreet社製3DCGコンテンツ作成ソフトウェア[6])ブラウザなど)に適する形に変換さる．これによりユーザは検索結果を視覚的に見ることができる．また非圧縮立体映像蓄積再生装置であるDVDA (Digital Video Disk Array)に転送することで，検索結果を立体映像として立体視する事も可能となる．

3. モデリングとスキーマ

DBMS上に仮想世界を構築する事を考える時，ユーザからの更新や問い合わせを想定したシステムを実現するためには，仮想世界をデータベースに蓄積するためのスキーマ定義，つまりは仮想世界のモデル化が必要となる．それに対し我々はontology層，mediator層，raw data層の3層からなるスキーマを定義することで，意味情報を仮想空間の物体の形状情報から分離して管理するモデル化手法を提案した[7].

3.1. ontology層

3次元仮想空間を構成するVRMLなどの従来の仮想空間記述方式を考えてみると，VRMLでは以下のように意味情報はオブジェクトの付加属性として表現されている．

```
Worldinfo{
        Title     “Box1.wrl”
        Info      [“物をしまう場所”]
        }
```

しかしこの手法では，付加情報が多岐にわたり，また意味情報も体系化されていないため，例えば”**収納する場所**”という検索に対し，”**Box1.wrl**”は意味的には同じでありながら属性値が異なるという理由から検索されないということになる．つまりこの手法では(1)構造を持った意味情報を記述する事はできない．(2)物が持つ意味が同じであるか異なっているかを**info**属性値内の文字列マッチングでしか表現できない．

またこの手法では意味情報に対する検索は**info**属性値の全検索となるため(3)モノの数が多くなった時計算に多大な時間を要する可能性がある，など問題点が多い．

これに対し本研究では，意味情報を仮想空間の物体の形状情報から分離し，意味情報の整理を行う．具体的には，仮想空間内に必要な意味情報を抽出し，意

味情報同士の関係を整理する事で，概念の明確な記述を行い，データベースに蓄積する．このような「概念の明示的な規約」は ontology[4][5]と呼ばれる．つまり本研究では，ontology に基づいて記述された意味情報を ontology 層に蓄積する．

3.2. raw data 層

Raw data 層では仮想世界を構成している物体のオリジナルの形状情報を蓄積しており，物体と物体の物理的な親子関係やポリゴンデータは全て raw data 層で管理される．そのため raw data 層に蓄積されるデータは，単なる形状データだけとする(つまりは，どのポリゴン群が「机の引出し」を表しているかのような意味情報は一切 raw data 層には含まれない)．この形状データは **Shape_object** クラスとして定義され，形状データが一般的に持つ「頂点リスト」「面ループ」「色」などが関連属性として保証されれば良い．またある形状データが「子部品群」の集約によって定義される事は一般によくある．このような親子関係と，そこから派生する相対座標，相対角度も含めて raw data 層に蓄積する．

3.3. mediator 層

ontology がもつ「意味」に基づいて仮想世界をモデリングした際の，仮想世界(raw data 層)に存在する個々の「もの」を mediator[8]として別個作成し，またその「もの」に対し raw data 層から近似して得られる簡略化されたポリゴン情報も共に mediator 層に蓄積する．つまり mediator とは，仮想世界から取得したオリジナルデータである raw data と，意味的な情報を規定した ontology との仲介者である．

また検索に対する mediator 導入の利点は 2 つ有り，1 つはオリジナルの形状情報を raw data 層に残したまま，mediator は簡略化されたポリゴンデータを持つことである．少ないポリゴンで表現できるという事は，物体同士の空間的な位置関係などを求める空間検索[9]に対し計算時間的に無理なく実現できる上に，検索結果を同一データベース上に存在する関連したオリジナルデータ(raw data)を表示する事で mediator の作成されていない物についても表示でき，フォトリアリスティックな CG を表示することができる．もう一つは，mediator を用いてポリゴンオブジェクトを統一されたモデルによって表現する事で，CG に対する選択演算，平均値や最大値などの集約演算を実現することである．

3.4. 3 層モデルを用いたスキーマ

以上のモデル化手法を用いて実際にオフィスルームをモデル化し，UML を用いてスキーマを記述した例を図 2 に示す．なお図 2 においてクラス間を結ぶ線は互いに参照可能であり，これにより例えば raw data 層のオリジナルの形状情報と，上位層の概念的な情報を相互に反映する事ができ，つまりは 3 層がそれぞれ反映しあう事ができている．

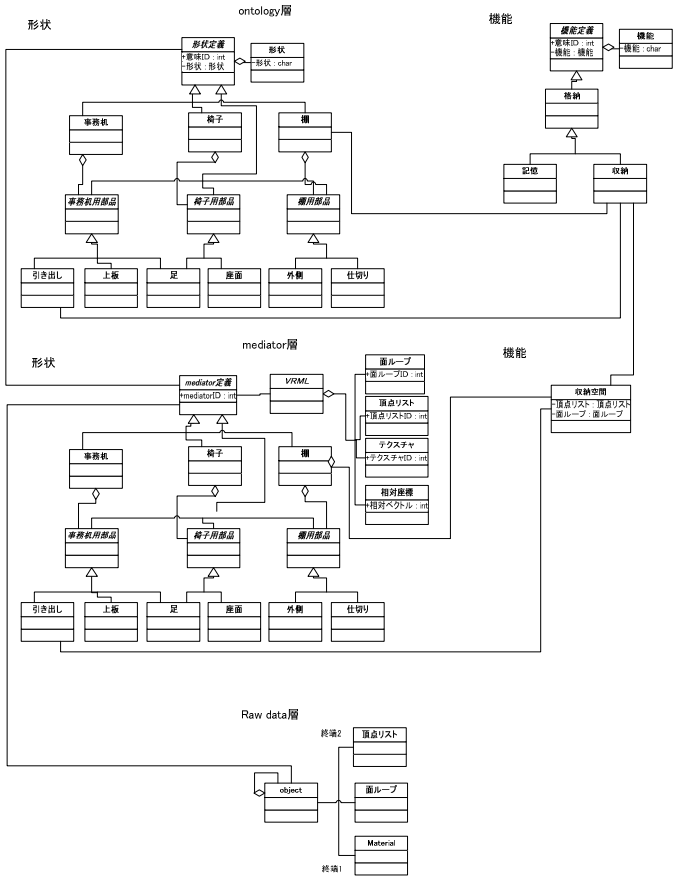

図 2.オフィスルームのスキーマ

4. SQLserver2000 を用いた実装と検証

本研究の目的は，3.で示したモデル化手法の有用性を示すために，実際に DBMS を用いてシステムを実装し評価することにある．よって一般的な機能を持つ DBMS であれば，システムの中核をなす DBserver の役割を実現できる戸考え，今回は 1 具体例として SQLserver2000 を用いて実装を行った．また，実装にあたり DBserver として使用したコンピュータは Dell Computer の Dell Precision530 (Xeon2.4GHz，実装メモリ 1GB)である．

4.1. RDBMS を用いた実装

今回本研究において RDBMS を用いた理由としては，フラットな構造による表現を目的としたためである．3.で示したように，本研究のデータベースにおいては，仮想空間を ontology 層，mediator 層，raw data 層の 3 層から捉えており，それらが互いに対等に参照し合うという構造をとっている．OODB はグラフィックや画像を取り扱うには向いているが，オブジェクトごとに上下構造を持たさずに双方向の参照やフラットな構造で情報を表現するためには工夫を要し，必ずしも適切であるとは言いがたい．本研究では，mediator による

意味(ontology)と実データ(raw data)の対応づけをデータモデリングの中核として実現し，その評価を行う事にある．そこで本論文では，できる限り汎用的な設計手法でフラットな構造による表現を望む観点から，RDBMS を用いて実装を試みた．

4.1.1. RDBMS でのスキーマ

本研究では 3 章で示したモデル化手法にのっとり，SQLserver2000 上に仮想空間を実装する事を試みる．3 章でしめしたスキーマ設計は， UML によりかかれた図 2 である．よって，SQLserver2000 は RelationalDBMS のため，図 2 で定義した UML で書かれたスキーマに対しリレーショナルモデルのスキーマに忠実に変換しなくてはならない．そこで以下の変換アルゴリズムに従って UML のスキーマをリレーショナルモデルのスキーマに変換した．(1)1 つのクラスに対し一つのテーブルを用意する．(2)shape_object とポリゴンデータ間のパス以外のパス に対しては，1 つのパスに対し 1 つのテーブルを用意する．

以上の方針で設計したリレーショナルモデルのスキーマ図を図 3～5 に示す．

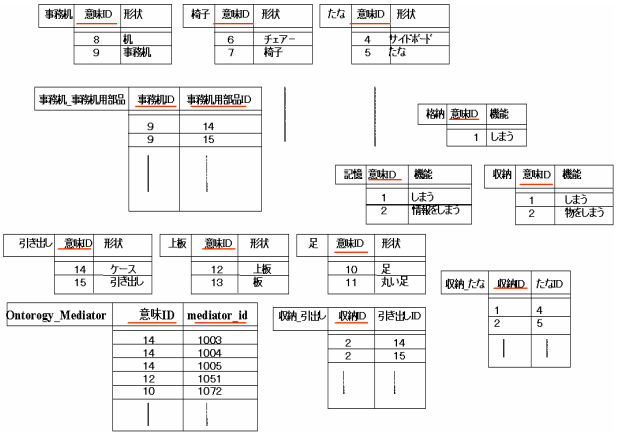

図 3. ontology 層

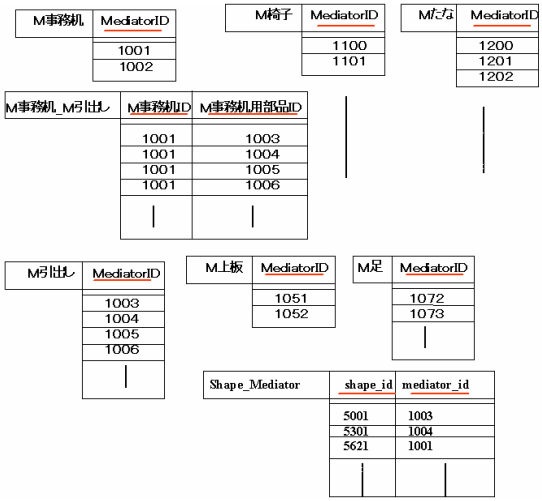

図 4. mediator 層

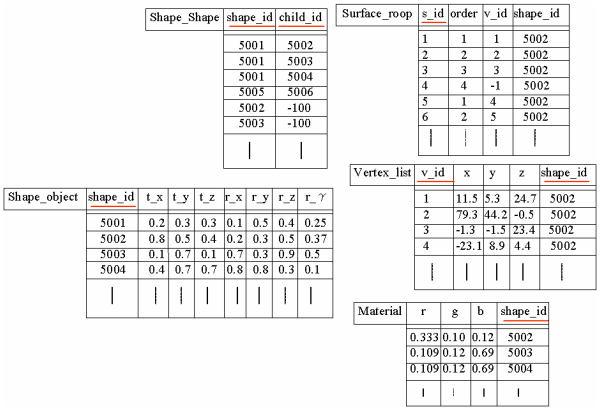

図 5. raw data 層

なお図 2，図 3～5，および(2)から分かるように，一つのパスはリレーショナルモデルのスキーマでは一つのテーブルとなっており，互いに参照可能の関係にある．これにより例えば raw data 層と mediator 層で言えば，図 4.のテーブル **Shape_mediator** により，raw data 層での物理的な情報と mediator 層の意味的な情報を相互に反映する事ができている．

4.1.2. インスタンスの挿入

図 3～5 に示したスキーマに対するインスタンスの挿入について説明する．仮想世界に対する Ontology 層，mediator 層のインスタンスは，仮想世界をスキーマにのっとった形でモデル化した時のインスタンスを蓄積する．Raw data 層のインスタンスは仮想世界に存在する膨大な物体の形状情報をまるごとスキーマに合う形で蓄積しなくてはならない．そこで本研究ではまず，実物の机を精密に測定し，3ds max をデザインツールとして作成した机をデータベースに蓄積する事を試みた．その机データを図 6 に，机データの形状情報を図 7 に示す．

データの蓄積方法としては，

(1)3ds max で作成された仮想物体を VRML 形式のテキストデータにエクスポートする．

(2)そのテキストに対して，リレーショナルモデルでのそれぞれのスキーマに必要なインスタンスを，文字列マッチングによる自動抽出プログラムによりそれぞれ CSV 形式のテキストファイルとして抜き出す．

(3)必要なインスタンスだけが書かれた CSV 形式のテキストファイルを SQLserver2000 にインポートする．

図 6. 3ds max で作成された机データ

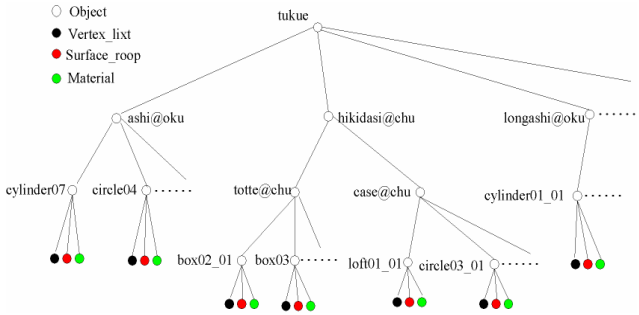

図 7. 机データの形状情報

以上により raw data 層のスキーマに則った形でデータベースにデータが実際に蓄積できた.

4.2. 「コミュニティ空間」に対する検索

次に，4.1.で構築されたデータベースに対し，ユーザからの検索に対するシステム内部での操作を，具体的に SQL を動かすことで検証する(以下の検索の対象は図 3～5 のインスタンスを用いている).

4.2.1. 検索の流れ

本節ではシステム内部での検索の流れを示すと共に，検索によって本システムの有用性を実際に例として 3 つの Query を行うことで考える.

(1) 意味世界に対する検索

Q1:物をしまうための引出しの種類を示せ.

ここでは ontology 層に対する選択 query を行う．具体的には図 3 で示される ontology 層のテーブル**収納**，テーブル**収納_引出し**，テーブル**引出し**に対する選択 query を行う．実際に実行した SQL とその検索結果を図 8 に示す.

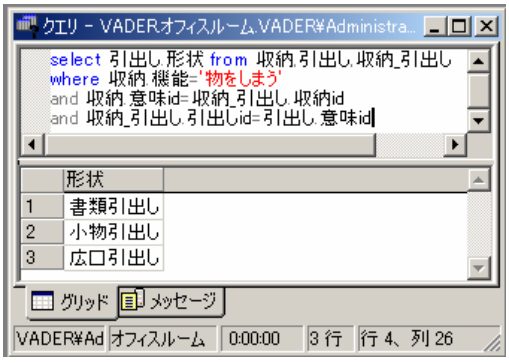

図 8.SQL「物をしまう_引出し」

これにより，この世界では“物をしまう”ための「引出し」は“書類引出し”,“小物引出し　”,“広口引出し”, の 3 種類存在することがわかる．これは意味世界において，意味間の関係に基づく検索が可能であることを示している.

(2)意味形状が対象世界に存在するかどうかに関する検索.

Q2:物をしまうための引出しと結び付けられている mediator を示せ.

ここでは mediator_id を示すことが Q2 の答えとなる．なぜなら mediator とは 3.3 節で述べた通り，ある意味を持つ物体が対象世界に個々の「もの」として存在することを表しているからである．実際に実行した SQL とその検索結果を図 9 に示す.

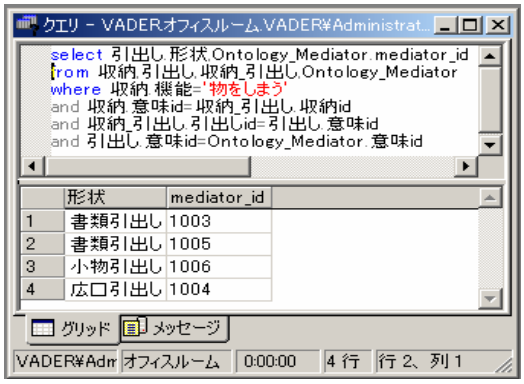

図 9.SQL「物をしまう_引出し_mediator」

これにより“物をしまう”ための「引出し」は“1003”,“1004”,“1005”,“1006”と，この世界の中で形を持って 4 個存在している，という事実を表す．これは，対象世界に存在する意味をもった「もの」に対する検索が可能であることを示している.

(2) 意味形状の，実際の形状データを求める検索

Q3:物をしまうための引出しと結び付けられている raw data を示せ.

ここでは，ontology 層，mediator 層，raw data 層をまたぐ宣言的検索を行う．そして mediator と結び付けられている raw data，すなわち shape_object を得る．実際に実行した SQL とその検索結果を図 10 に示す.

図 10.SQL「物をしまう_引出し_mediator_shape」

これにより，“物をしまう”ための「引出し」はこの世界の中で形を持って存在し，その形状データは“hikidasi@sita”,“hikidasi@ue”,“hikidasi@chu”,“BIG__hikidasi”の 4 つにそれぞれ関連付けられていることがわかる．これは，意味情報をもたない raw data に対し意味に基づく検索が可能であることを示している．実際の形状データの CG 表示に関しては次節でふ

れる．

また本システムでは RDBMS を用いたフラットなデータ構造を取っているので，例えば mediator 層から ontology 層をたどるなど，上下構造を持たずに双方向の検索が可能となる．

4.2.2. シーングラフの作成

本研究では，DB に対する検索結果を CG によってユーザに表示する．そのために検索結果として XML で記述されたシーングラフを生成する．本研究で生成される具体的なシーングラフの例を図 12.に示す．

図から分かるように，シーングラフは階層構造で示されており，また必要なタグの間に値が埋め込まれている．本研究では RDBMS を用いてシステムを実装しているため，raw data 層の階層構造は図 5 のテーブル **shape_shape** で示される親子関係を用いて表現されている．よってシーングラフを作成する際，raw data 層に対する検索は SQL を複数回発行し，複数枚のテーブルとして取り出され，必要なタグをつけてシーングラフに埋め込む形をとる．

ここで本システムでのシーングラフ作成アルゴリズムを図 13.に示す．

ここで@current は shape_object の shape_id が入る変数，@level は shape_object の階層の高さが入る変数，そして T は shape_object の transform が入る変数である．また S は検索により raw data 層に最初に与えられた shape_object の初期ノード集合でありであり，例えば図 10 の検索で言えば shape_object「**hikidasi@sita**」，shape_object「**hikidasi@ue**」，shape_object「**hikidasi@chu**」，shape_object「BIG__hikidasi」の 4 つがこれにあたる．C は shape_object とその階層の高さが入る集合であり，本アルゴリズムでは一時的な集合として用いた．G は検索結果の出力先となるシーングラフである．D は shape_object が入る集合であり，

```
<world>
  <walk-pass></walk-pass>
  <material></material>
  <SubObject>
    <Object>
      <id>tukue01</id>
      <transform>
        <translation>1 1 1</translation>
        <rotation>1 0 0 10</rotation>
      </transform>
      <LinktoMediator>33</LinktoMediator>
      <subobject>
        <object>
         <id>ashi01</id>
         <material>
            <color>1 0 0</color>
          </material>
         <transform>
            <translation>1 4 5</translation>
                    .
                    .
```

図 12.シーングラフの例

```
1: initially @current, @level, @line,T, S, C={ϕ,ϕ}, G, D={ϕ}
2: for each element s ∈ S do
3:     insert s into D
4:     while element d ∈ D exists do
5:         if exists(parent(d)) then
6:             T=mat(parent_t(d))
7:             s.trans=s.trans*T
8:             insert parent(d) into D
9:             remove d from D
10:        end if
11:    end while
12:    insert {s,1} into C
13:    @level=1
14:    while @level>0
15:        if exists( c.level=@level ) then
16:            @current=c.node
17:            append "<object>" into G
18:            insert tag(@current) into G
19:            remove c from C
20:            if exists(child(@current)) then
21:                insert {{child(@current)}*{@level+1}} into C
22:                @level=@level+1
23:                append "<subobject>" into G
24:            end if
25:            else then
26:                insert vertex(@current) into G
27:                insert material(@current) into G
28:                append "<object>" into G
29:            end else
30:        end if
31:        else then
32:            @level=@level-1
33:            if(@level!=0) then
34:                append "<object>" into G
35:            end if
36:            append "<subobject>" into G
37:        end else
38:    end while
39: end for
40: return G
```

図 13.シーングラフ作成アルゴリズ

こちらも一時的な集合として用いた．

このアルゴリズムでは，実際にデータベースに対し複数回の SQL を発行し，複数枚のテーブルを得て，シーングラフに値を埋め込んでいっているのがわかる．具体的にはまず **raw data** 層の **shape_object** に対し，一つづつ上にも **shape_object**(親ノード)を持つか聞いて行き(5 行目)，あったらその **transform** を自分の **transform** に座標系変換を行う事で反映させ(6～9 行目)，無くなったら処理をやめる(4～11 行目のループ)．

そして次に一つづつ下に **shape_object**(子ノード)を持つか確認し(20 行目のループ)，有ったらその **shape_object** 群を保持し(15 行目の if 内)，無かったらそのポリゴンデータを保持する(25 行目の else 内)．そして保持されたデータをシーングラフに見合う形でタグを付けて埋め込んで行く(18 行目，23 行目と，34，36 行目)．

またアルゴリズムから分かる通り，raw data 層の形状情報はリーフノードのみがポリゴンデータを持っており，その他の中間ノードは **shape_id**，**mediator_id**，**transform** だけを持つ．

ここで，図 13.において使用される関数はそれぞれ一つの SQL に対応している．その関数と SQL の対応

関係を図 14 に示す．また図 14 の各関数の機能を表 1 にまとめる．

表 1.の parent_t()に関しては，rotation も取るのが理想だが，今回のケースでは rotation の変換は必要なかったので除外した．この点に関しては今後の課題とする．

表 1.各関数の機能

関数名	引数	機能
child()	shape_id	shape_object の子ノードを返す
vertex()	shape_id	shape_object の頂点リストを返す
material()	shape_id	shape_object のマテリアルを返す
tag()	shape_id	shape_object の shape_id，mediator_id transform を返す
parent()	shape_id	shape_object の親ノードを返す
parent_t()	shape_id	shape-object の親ノードが持つ translation

```
  /* 関数 child(@current) */
select child_id
from shape_shape
where shape_id='@current'

/* 関数   vertex(@current) */
select vertex_list.x,vertex_list.y,vertex_list.z,surface_roop.orders
from vertex_list,surface_roop
where vertex_list.shape_id='@current'
and vertex_list.shape_id=surface_roop.shape_id
and surface_roop.v_id=vertex_list.v_id
and surface_roop.v_id!=-1
order by surface_roop.s_id

/* 関数   material(@current) */
select r,g,b
from Material
  where shape_id='@current'

  /* 関数   tag(@current) */
  select shape_object.shape_id,shape_object.t_x,shape_object.t_y,shape_object.t_z,
       shape_object.r_x,shape_object.r_y,shape_object.r_z,shape_object.r_y,
       shape_mediator.mediator_id
from shape_object,shape_shape,shape_mediator
where   shape_shape.shape_id='c.node'
and shape_shape.child_id=shape_object.shape_id
and shape_object.shape_id=shape_mediator.shape_id

/* 関数   parent(d) */
select shape_id
from 仮 shape_shape
where child_id='d'

/* 関数   parent_t(d) */
select shape_object.t_x,shape_object.t_y,shape_object.t_z,
       shape_object.r_x,shape_object.r_y,shape_object.r_z,shape_object.r_y
from shape_object,shape_shape
where shape_shape.child_id='d'
  and shape_shape.shape_id=shape_object.shape_id
```

図 14.関数と SQL

4.2.3. 検索結果の CG 表示

以上のアルゴリズムを基に，VisualBasic.NET を用いてシーングラフ作成の自動化を行った．実際に図 10 の SQL に対する検索結果をシーングラフ作成まで行い，そのシーングラフを OpenGL を用いて実装した本システムのブラウザによって表示した様子を図 15 に示す．

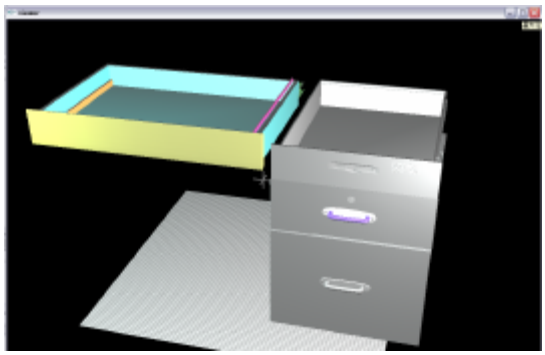

図 15.検索結果の CG 表示

これにより“物をしまう”ための「引出し」は，この世界の中で形を持って 4 個存在し，その形状データはメディエータと関連づけられた意味を持った CG としてユーザに視覚的に提供されており，この一連の流れがシステムとして実装されている事が示された．つまりは ontology 層に対する意味に関する検索により，検索結果が CG として得られた．また，この一連の操作は自動化されている．

4.3. 「コミュニティ」空間に対する更新

ユーザからの更新については，様々なものが考えられるが，今回は OpenGL ブラウザで，仮想物体をつかんで動かす picking によるオブジェクトの移動の更新について行う．

4.3.1. picking

OpenGL において仮想世界に存在する物体をつかんで動かす事を picking といい，物体として選択することを表す．picking に関しては，あるユーザが OpenGL ブラウザで物体を移動した際，その物体の移動後の座標をデータベースに更新する事で，その更新が「コミュニティ」空間に存在する全てのユーザのブラウザに対して反映する事ができるようになる．

本研究では，OpenGL ブラウザで行われた picking により移動した物体の，**mediator_id** と移動後の相対座標を，まずデータベース中の同じ **mediator_id** を持つ物の相対座標を更新し，次にその **mediator_id** に対応する raw data 層の shape_object の相対座標の更新を行う事でデータベースに反映させる．

具体的には，**mediator_id=ID** の物体を移動して，移動後の相対座標が **X**，**Y**，**Z** だった時，データベース中の mediator 層において **mediator_id** が **ID** の物の相対座標を **X**，**Y**，**Z** に置き換え，対応する shape_object の相対座標も **X**，**Y**，**Z** に置き換えた．Shape_object に

対する更新の際に発行する SQL を図 16 に示す.

```
/* 関数 picking(ID,X,Y,Z) */
update shape_object set
t_x=X,t_y=Y,t_z=Z
where shape_id='ID'
```

図 16.picking の際の raw data 層に対する update 句

なお，今回はデータベースにおいて，移動された物体の親ノードが mediator，raw data 共に一致しているため同じ相対座標値でそのまま更新できた．しかし実際は mediator 層と raw data 層の階層は必ずしも一致しているとは限らない．この点に対しては，mediator 層と raw data 層で相対座標の変換を行えば整合性を持って更新する事が可能となるが，この点に関しては今後の課題とする．

4.4. 高速ネットワークを用いた質問処理時間のベンチマーク

本システムの「コミュニティ」としての利用の可能性を検証するために，高速ネットワークを用いて質問処理にかかる処理時間のベンチマークを行った．

4.4.1. 実験方法

本節では，検索結果のポリゴン数とコミュニティのユーザ数に対する，質問処理に要する時間のベンチマークを行った．具体的には(1)データベース内に複数の SQL が発行される，(2)検索結果として複数枚のテーブルがユーザに届く，(3)複数枚のテーブルからシーングラフが作成される，までの一連の時間をユーザ数とポリゴン数を変えて測定した．なお使用したネットワークは伝送速度 1Gbps の高速ネットワークである．またユーザが用いたコンピュータはどれも CPU が pentium4 2.4GHz，実装メモリ 512MB である．

4.4.2. 実験結果

実験結果を図 17 に示した．これより 20000 ポリゴン(引出し 1 個分)程度ではユーザ数を増やしても時間的に無理なく利用できるが，それ以上では「コミュニティ」としては多大な処理時間を要してしまった．これに対してはシーングラフ作成アルゴリズムを改良することで，処理時間の改善を目指す．

5. まとめと今後の課題

本稿では，「コミュニティ」空間の構築において，本研究で掲げるモデル化手法の実用性を，実際に SQLserver2000 をエンジンとしてシステムを作成する事で実証した．

具体的には実際に仮想空間をデータベースに構築

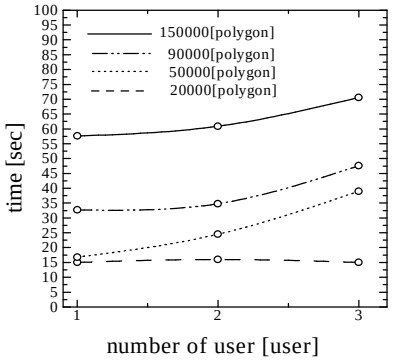

図 17.ユーザ数，ポリゴン数に対する処理時間

し，データベースに対する検索，更新に要する時間のベンチマークを行った．今後の課題としてはユーザからの更新に対するシステム内部での操作方法のさらなる検証，シーングラフ作成アルゴリズム改善による検索に要する時間の短縮,仮想世界の増大,「コミュニティ」空間に対する複数クライアントの同時実行制御などを考えていく．

謝辞 本研究の一部は文部科学省科学研究費補助金(若手研究(B)、課題番号 40313473)および、財団法人横浜工業会研究助成による。また実装にあたり、「日本データベース学会・マイクロソフト株式会社共催 2002 年度データベース研究支援プログラム」の支援を得た。ここに記して謝意を表する。

文　献

[1] 水上孝一，“コンピュータグラフィックス，”朝倉書店.

[2] T. Tomii, K. Salev, S. Imai, and H. ArisawaIgeta, “Human Modelling and Design of Spatio-Temporal Queries on 3D Video Database, ” Visual Database Systems 4 (VDB4), L’Aquila, Italy, 27-29 May 1998.

[3] 渡辺知恵美，増永良文：仮想世界データベースシステムにおけるマルチモーダル問合せ言語の設計に向けて，情報処理学会データベースシステム研究会報告，*Vol125，No.71, pp.289-296 2001*

[4] JOHN F. SOWA, ”KnowledgeRepresentation: Logical, Philosophical,and Computational Foundations”, Brooks/Cole,2000

[5] Gruber, Thomas R. ”A translation approach to portable ontology specifications. In Knowledge Acquisition, vol.5, pp199-220, 1993.

[6] http://www.discreet.com/products/3dsmax/

[7] 岡田直也，富井尚志 “ontology を用いた空間形状データの意味情報モデリング” データ工学ワークショップ, Mar, 2003

[8] 富井尚志，小林みなこ，有澤博“ 仮想 CG 空間へのマッピングによる現実シーンデータベースの設計 ” 電子情報通信学会論文誌，D-1，Vol.J82-D-I, No.1, pp.211-222, jan, 1999

[9] C. Chen, C. Zaniolo, “SQLst: A Spatio-Temporal Data Model and Query Language, ” ER2000 Conference, LNCS 1920,pp.96-111，Springer-Verlag, Berlin, Heidelberg, Amsterdam, 2000.