

いま、土木業界で 足りない人材とは？

今後必須となる「建設ICT」では
どのような人材・スキルが要求されるのかについて

2020年6月24日

大阪大学 大学院工学研究科

環境エネルギー工学専攻

准教授 福田 知弘

第4次産業革命へ

第1次産業革命

18世紀末以降の水力や蒸気機関による工場の機械化

第2次産業革命

20世紀初頭の分業に基づく電力を用いた大量生産

第3次産業革命

1970年代初頭からの電子工学や情報技術を用いた一層のオートメーション化

第4次産業革命

超スマート社会。IoT、ビッグデータ、AI、ロボット

1. 大量生産・画一的サービス提供から個々にカスタマイズされた生産・サービスの提供
2. 既に存在している資源・資産の効率的な活用
3. AIやロボットにより、従来人間によって行われていた労働の補助・代替などが可能

育成すべき資質・能力の三つの柱

学びに向かう力
人間性等

どのように社会・世界と関わり、
よりよい人生を送るか

国際感覚

「確かな学力」「健やかな体」「豊かな心」を
総合的にとらえて構造化

何を理解しているか
何ができるか

知識・技能

情報基礎

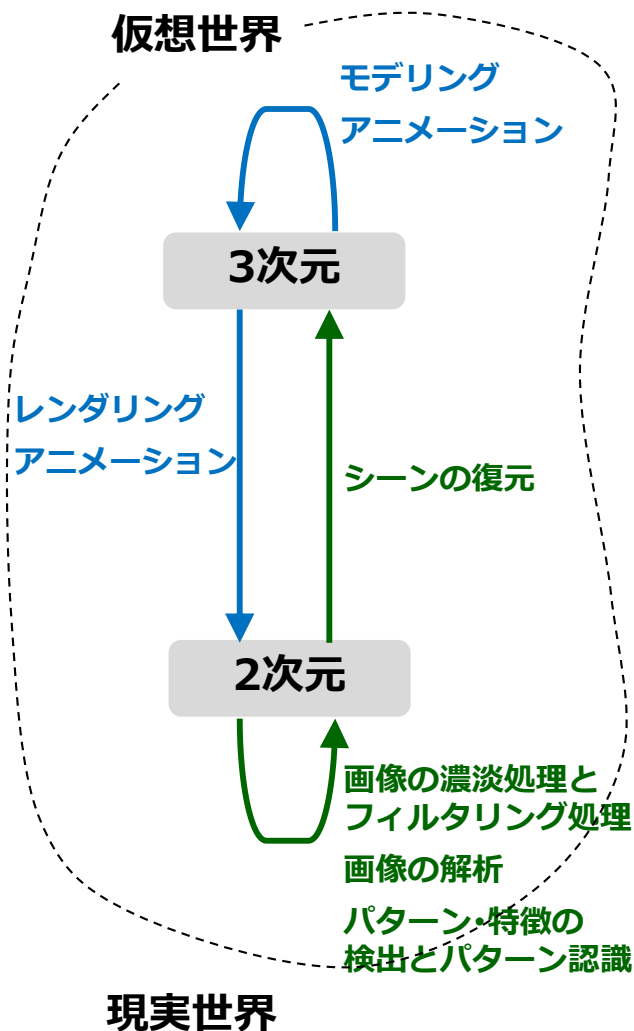
理解していること・できる
ことをどう使うか

思考力・判断力・表現力等

情報基礎

ビジュアル情報処理

現実世界



CG (Computer Graphics)

モデリング (modeling) : (これから作ろうとする) 現実世界にある3次元の物体の形状などの情報をコンピュータ内に表現し、操作できるようにするしくみ

レンダリング (rendering) : モデリングされた情報をディスプレイモニタなどに表示できる2次元の画像に変換する処理

アニメーション (amination) : 時間に伴う対象世界の変化をモデリングとレンダリングの両面から扱う技術

(広義の) 画像処理

画像の濃淡処理とフィルタリング処理 : 2次元の画像に変換された情報は、さらに濃淡処理やフィルタリング処理を用いて加工し、目的に合わせた新しい画像を作成

画像の解析 : 画像から有益な情報を抽出するための手法

パターン・特徴の検出とパターン認識 : 画像から特定のパターンや特徴を検出し、過去の学習結果と照合して、新たな画像を識別するパターン認識手法

シーンの復元 : 対象世界の情報を-CGの処理の流れとは逆に-対応する画像から復元する作業 (= **コンピュータビジョン CV** (Computer Vision))

ビジュアル情報処理の例 -1

現実世界

仮想世界

モデリング
アニメーション

3次元

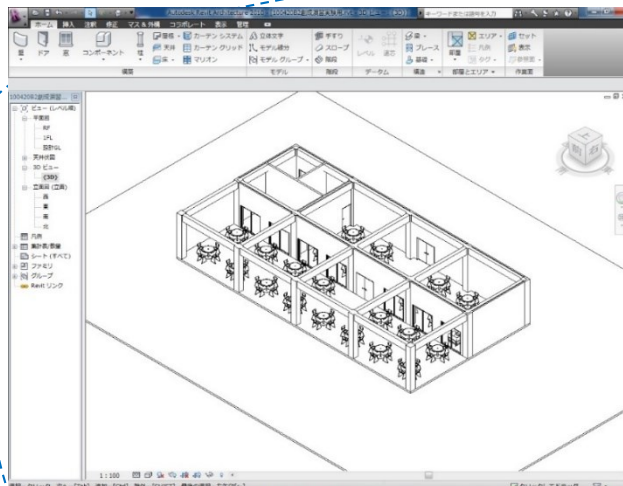
レンダリング
アニメーション

シーンの復元

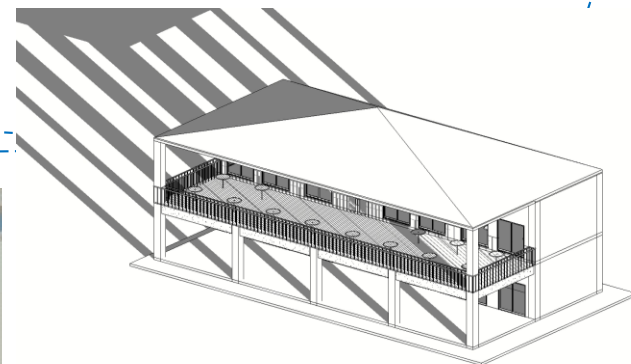
2次元

画像の濃淡処理と
フィルタリング処理
画像の解析
パターン・特徴の
検出とパターン認識

現実世界



BIMを用いたモデリング



日影シミュレーション



アイレベルからのレンダリング

ビジュアル情報処理の例 -2

現実世界

仮想世界

モデリング
アニメーション

3次元

レンダリング
アニメーション

シーンの復元

2次元

画像の濃淡処理と
フィルタリング処理

画像の解析

パターン・特徴の
検出とパターン認識

現実世界



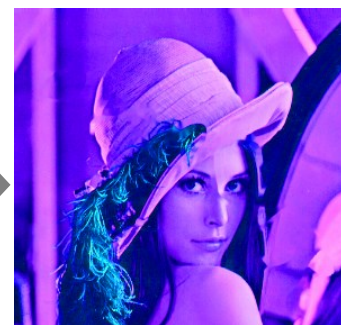
多地点から撮影した写真



三次元形状・色の復元（点群）



撮影された写真



加工



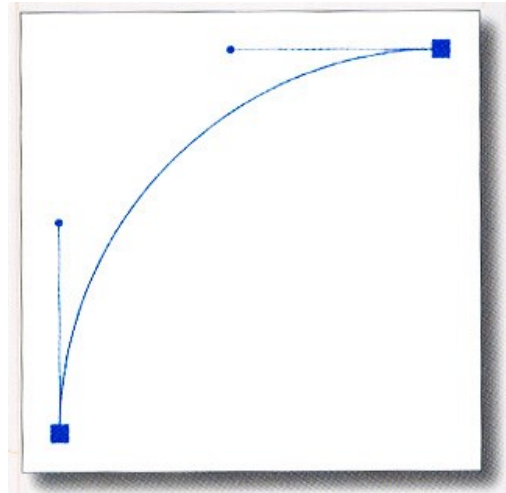
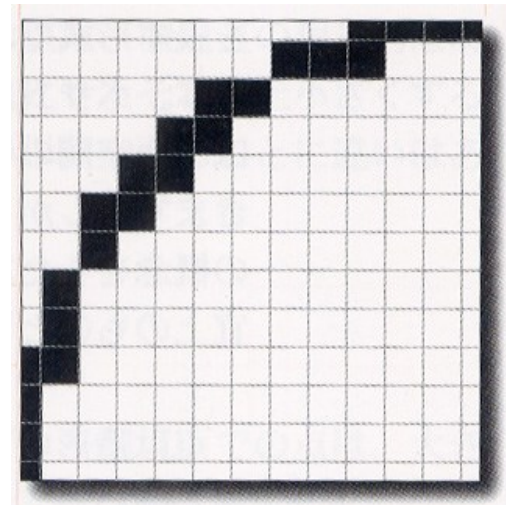
バス
BS01 HM
...

基本単位

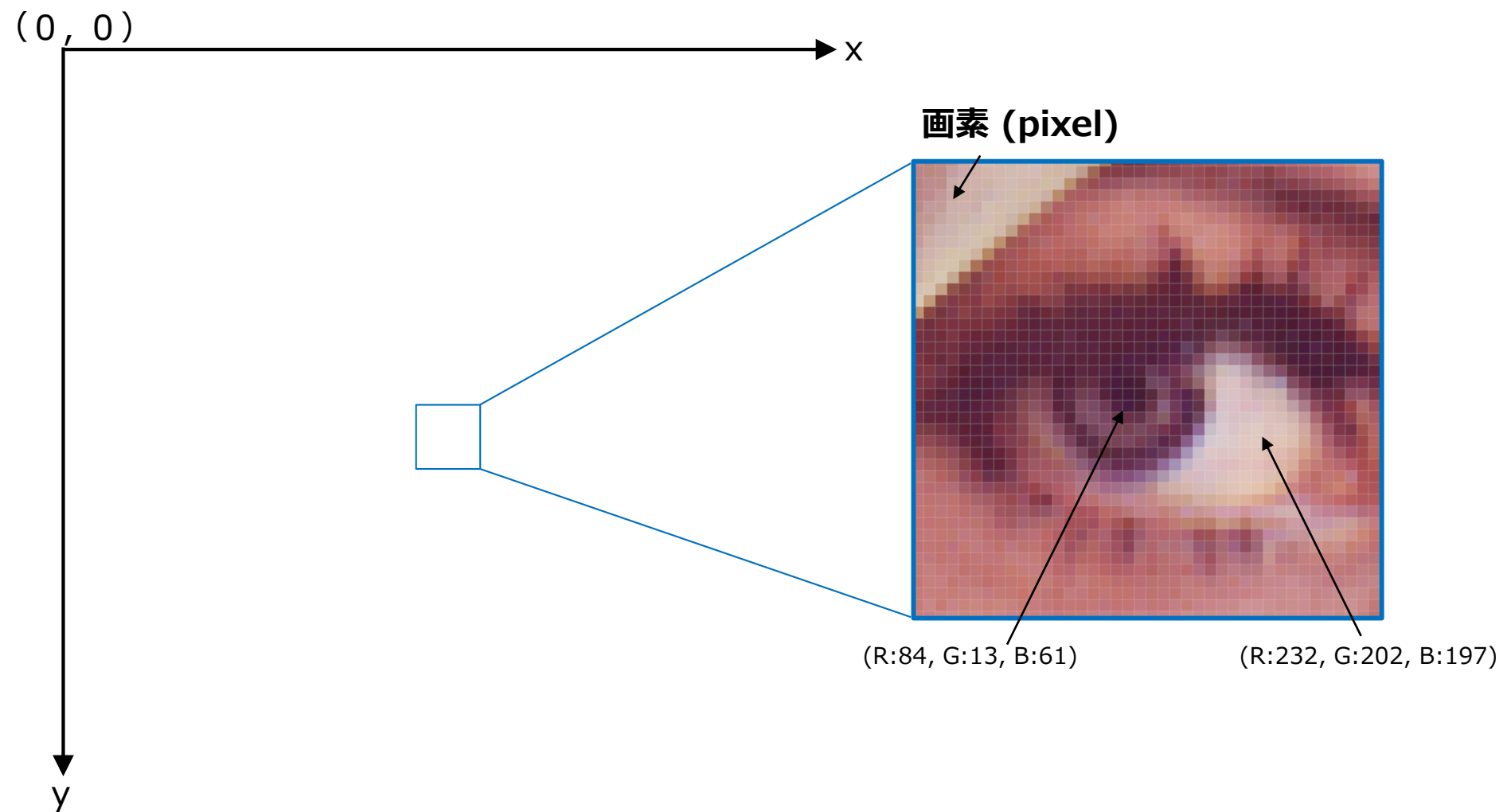
- データ最小単位 : bit
 - 1 byte = 8 bit
- データ転送レート : bps (bit per sec)
- 画像最小単位 : pixel
 - フルカラーの非圧縮画像データ量 : $24\text{bit/pixel} \times \text{pixel数}$
- 動画単位時間当たりのフレーム数 : fps (frames/sec)
 - フルカラーの非圧縮動画データ量 : $24\text{bit/pixel} \times \text{pixel数} \times \text{fps} \times \text{sec}$

図形・画像のデータ表現形式

- ラスタ型 (raster)
 - 小さな正方形の色面の集まりにより、視覚的イメージ (画像など) を表す
 - 画素 (ピクセル) の並びで面を構成するデータ構造
- ベクター型 (vector)
 - 図形のアウトライン (輪郭線) によって視覚的イメージを表す形式
 - ベクトルの集合として扱う
 - 幾何学的変換 (平行移動、回転、拡大縮小など) を高い精度で容易におこなえる



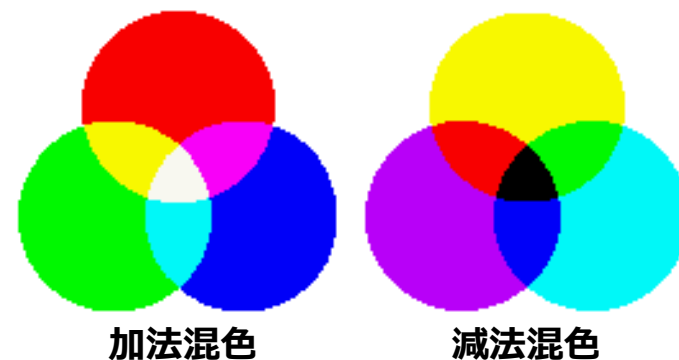
デジタル画像（ラスタ型）：画像座標系, 画素, RGB



光の三原色・色の三原色と混色

・ 色光（色をもつ光）の三原色

- ・ 黄みの赤・緑・紫みの青（RGB）
- ・ 混色すると明度が上がる（白に近づく）ので加法混色（additive color mixture）
- ・ （例）ディスプレイ

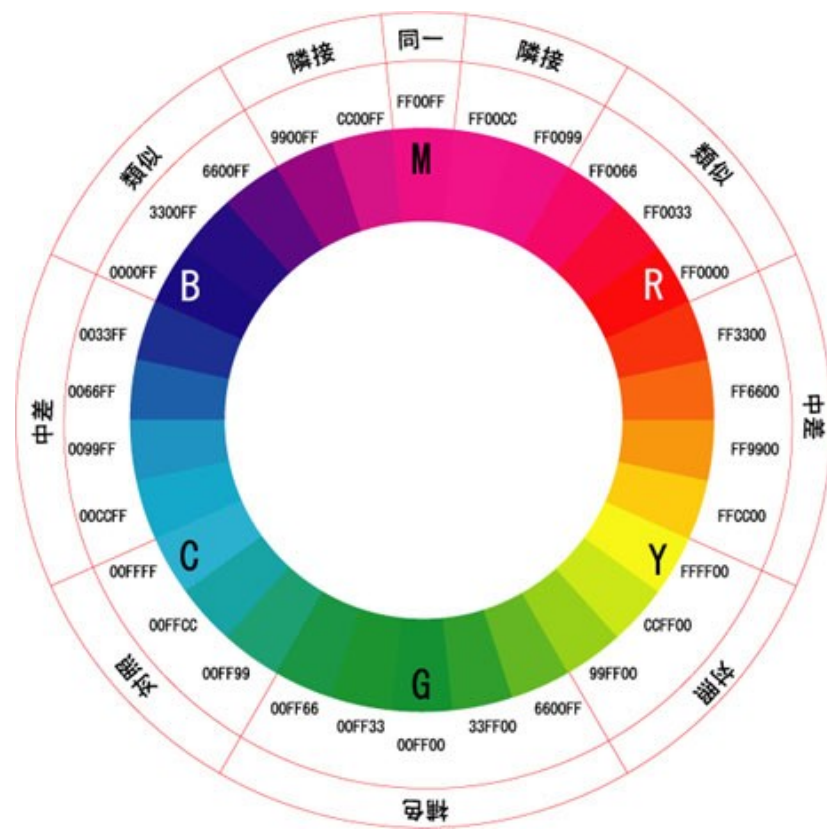


・ 色料（色をもつ材料）の三原色

- ・ シアン（緑みの青）・マゼンタ（赤紫）・イエロー（黄）（CMY）
- ・ 混色すると明度が下がる（黒に近づく）ので減法混色（subtractive color mixture）
- ・ （例）紙の印刷

・ 原色

- ・ どんな色を混ぜ合わせてもつくりができない色



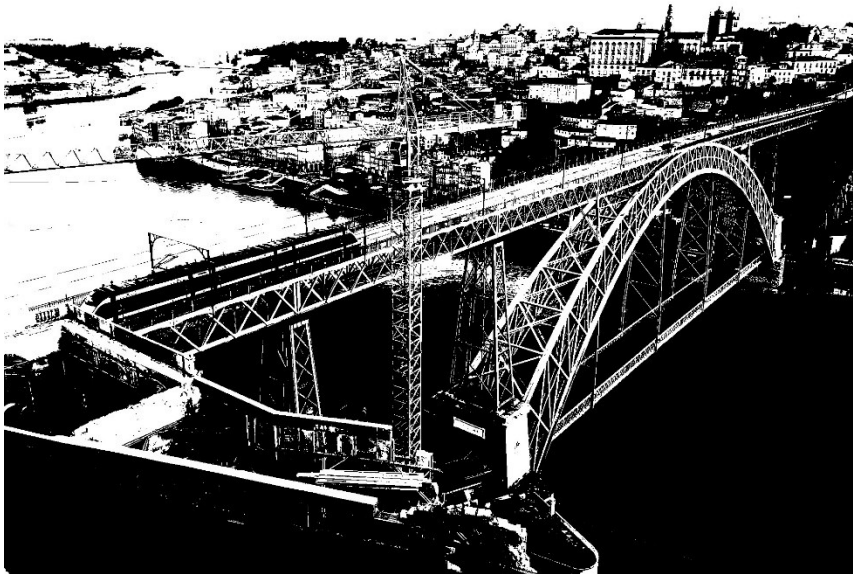
デジタル画像（ラスタ型）の表現例 -1

- 1pixelあたりの情報量が1bitのとき

- 色彩は白と黒の2階調

データ最小単位

- Binary digit (2進数字) の略 0 1
- 通常、1B (byte) = 8 bit



ポスタリゼーション



ハーフトーンスクリーン

1bit画像イメージ

デジタル画像（ラスタ型）の表現例 -2

- 1pixelあたりの情報量が8bitのとき
 - $2^8=256$ 種類の情報を扱える



グレースケール grayscale image
=256階調

8bit画像イメージ

デジタル画像（ラスタ型）の表現例 -3

- 1pixelあたりの情報量が24bitのとき
 - Red $2^8=256$ 階調、Green $2^8=256$ 階調、Blue $2^8=256$ 階調の色光の組み合わせによる約1600万色の色彩表現が可能（RGBカラー画像：RGB color image）
 - $256 \times 256 \times 256 = 2^{24}=16,777,216$ 色
 - 色深度（色彩表現のための色数）が増えるほど自然な表現が可能



24bit画像イメージ

練習問題 -1 (7分)

以下のようなグレースケールの非圧縮画像がある。

この画像のデータ量を求めなさい。



3024 pixel

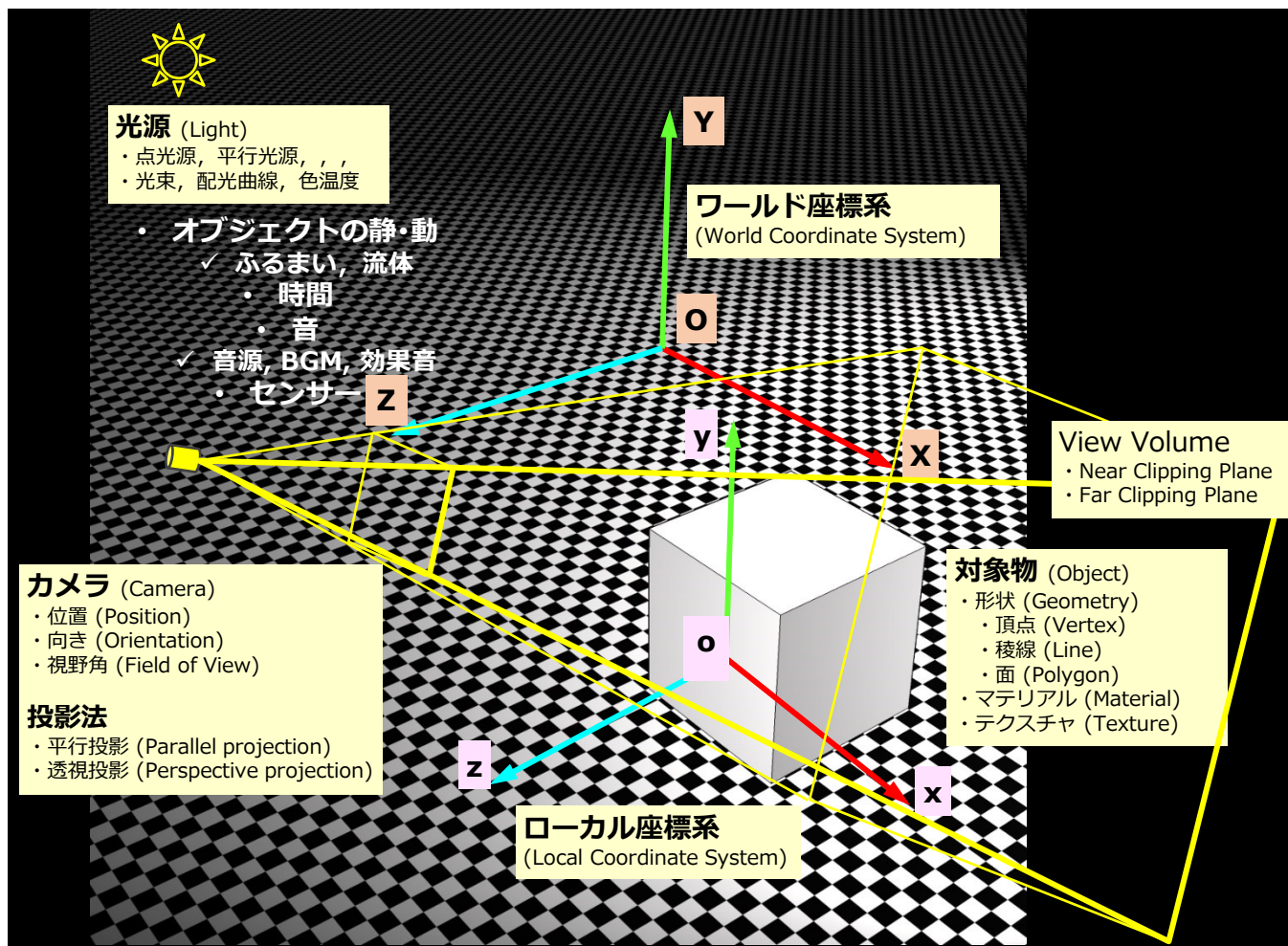
4032 pixel

3次元仮想空間記述言語：VRML97 (現・web3D)

- **HTML** (Hyper Text Markup Language)
 - 文字のフォント、大きさ、色などの表現、画像、アンカー（リンク）、表などをタグで設定しながらWebサイトページを作成できる。
 - HTML対応のブラウザにより、インターネット上で誰もが閲覧できる。テキスト（アスキー）で仕様に則り記述する。HTMLを作成するソフトもある。
- **VRML** (Virtual Reality Modeling Language)
 - 限定されたOSやソフトに依存することなく、VRML対応のブラウザにより、インターネット上の3次元CGをリアルタイム・レンダリングにより閲覧できる。
 - 3次元表示された仮想オブジェクトや仮想空間を様々な角度から眺めたり、アフィン変換（平行移動、回転、拡大縮小）したり、3次元仮想空間をウォークスルーしたりできる。また、他のHTMLをアンカー機能として3次元空間内に配置することができる。いわば、HTMLの3次元版。拡張子は .wrl
 - VRMLファイルは、テキスト（アスキー）で仕様に則って記述する。3DCAD/CGから出力することもできる。基本的には、テキストエディタとVRML対応のブラウザがあれば誰もがVRMLで3次元オブジェクトを容易に制作できる。
 - 現状、他のゲームエンジンと比較すると、機能は少ないものの、基本的な機能を有している。記述構造の理解がしやすいため、本研修で採用する。

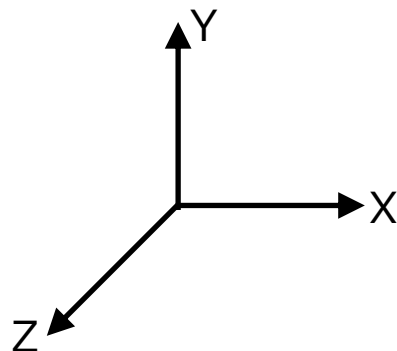
3次元仮想空間記述言語：VRML97

- VRMLの中には、座標やポリゴンで表される形状データ、プリミティブ（基本図形）、マテリアル（質感、材料の属性）、テクスチャ、光源、視点情報などの3次元CG情報やWebの機能が含まれている。VRML97では、音楽、センサー、オブジェクトのアニメーション機能などにより、インタラクティブ（interactive）な表現が可能である。
 - これらの各情報は、ノード（node）というVRMLの仮想空間上の立体情報を記述する単位（オブジェクトと考えてよい）ごとにグルーピングされ、階層的なツリー状のデータ構造でまとめられている。

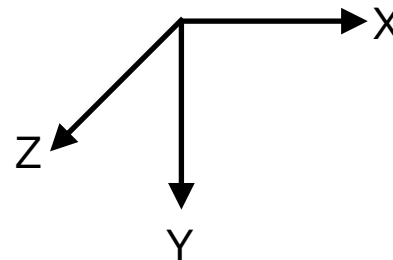


3次元座標系 (3-Dimensional Coordinate System)

- 3次元直交座標系 (three-dimensional orthogonal coordinate system) は、x軸、y軸に対して、z軸がどの向きに設定されているかにより、右手系 (right-handed coordinate system) と左手系 (left-handed coordinate system) に分けられる。
- 右手系or左手系、どの軸を上向きにするかは分野や用途によって様々。CGでは、特定の座標系にこだわるのではなく、それぞれの処理に便利な座標系を使い分け、必要に応じて座標系間で座標値を変換する。
- 他からのデータを受領して、自らのシステムに入力する時、座標系と長さの単位 (メートル、インチ) に注意。



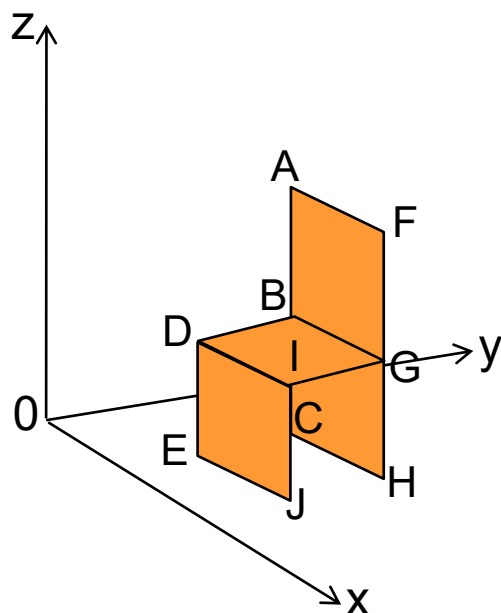
OpenGL



Processing

簡単なモデリング

- 3次元図形の形状を数値的に記述することをモデリング (modeling) と呼び、記述される形状のデータを形状モデル (geometric model) と呼ぶ。
- 多角形 (polygon) を定義する際、頂点 (vertex) の座標とその順序で表すことができる。
- 頂点は何個用いても構わないが、4個以上用いた場合に平面上にあるとは限らないので、一般には頂点が3個の三角形が用いられる。
- 多角形の集まりで表された形状データは、メッシュ (mesh) と呼ばれることもある



頂点の座標

A (2, 5, 9)

B (2, 5, 5)

...

J (5, 2, 1)

ポリゴン (頂点の列で表す)

A-C-H-F

B-D-I-G

D-E-J-I

Node Reference

[6.1 Introduction](#)

[6.2 Anchor](#)

[6.3 Appearance](#)

[6.4 AudioClip](#)

[6.5 Background](#)

[6.6 Billboard](#)

[6.7 Box](#)

[6.8 Collision](#)

[6.9 Color](#)

[6.10 ColorInterpolator](#)

[6.11 Cone](#)

[6.12 Coordinate](#)

[6.13 CoordinateInterpolator](#)

[6.14 Cylinder](#)

[6.15 CylinderSensor](#)

[6.16 DirectionalLight](#)

[6.17 ElevationGrid](#)

[6.18 Extrusion](#)

[6.19 Fog](#)

[6.20 FontStyle](#)

[6.21 Group](#)

[6.22 ImageTexture](#)

[6.23 IndexedFaceSet](#)

[6.24 IndexedLineSet](#)

[6.25 Inline](#)

[6.26 LOD](#)

[6.27 Material](#)

[6.28 MovieTexture](#)

[6.29 NavigationInfo](#)

[6.30 Normal](#)

[6.31 NormalInterpolator](#)

[6.32 OrientationInterpolator](#)

[6.33 PixelTexture](#)

[6.34 PlaneSensor](#)

[6.35 PointLight](#)

[6.36 PointSet](#)

[6.37 PositionInterpolator](#)

[6.38 ProximitySensor](#)

[6.39 ScalarInterpolator](#)

[6.40 Script](#)

[6.41 Shape](#)

[6.42 Sound](#)

[6.43 Sphere](#)

[6.44 SphereSensor](#)

[6.45 SpotLight](#)

[6.46 Switch](#)

[6.47 Text](#)

[6.48 TextureCoordinate](#)

[6.49 TextureTransform](#)

[6.50 TimeSensor](#)

[6.51 TouchSensor](#)

[6.52 Transform](#)

[6.53 Viewpoint](#)

[6.54 VisibilitySensor](#)

[6.55 WorldInfo](#)

データ構造

あるNodeは、他のNode
を子に持つことができる

Node Type	Field	Valid Node Types for Field
Anchor	<i>children</i>	Valid children nodes
Appearance	<i>material</i>	Material
	<i>texture</i>	ImageTexture, MovieTexture, Pixel Texture
Billboard	<i>children</i>	Valid children nodes
Collision	<i>children</i>	Valid children nodes
ElevationGrid	<i>color</i>	Color
	<i>normal</i>	Normal
	<i>texCoord</i>	TextureCoordinate
Group	<i>children</i>	Valid children nodes
IndexedFaceSet	<i>color</i>	Color
	<i>coord</i>	Coordinate
	<i>normal</i>	Normal
	<i>texCoord</i>	TextureCoordinate
IndexedLineSet	<i>color</i>	Color
	<i>coord</i>	Coordinate
LOD	<i>level</i>	Valid children nodes
Shape	<i>appearance</i>	Appearance
	<i>geometry</i>	Box, Cone, Cylinder, ElevationGrid, Extrusion, IndexedFaceSet, IndexedLineSet, PointSet, Sphere, Text
Sound	<i>source</i>	AudioClip, MovieTexture
Switch	<i>choice</i>	Valid children nodes
Text	<i>fontStyle</i>	FontStyle
Transform	<i>children</i>	Valid children nodes

VRML97: Node と Field

6.53 Viewpoint

“DEF 名前 Node名” でノードのオブジェクト定義

Node
Field

```
Viewpoint {  
  eventIn      SFBool      set bind  
  exposedField SFFloat      fieldOfView 0.785398 # (0,  $\pi$ )  
  exposedField SFBool      jump TRUE  
  exposedField SFRotation  orientation 0 0 1 0 # [-1, 1], (- $\infty$ ,  $\infty$ )  
  exposedField SFVec3f     position 0 0 10 # (- $\infty$ ,  $\infty$ )  
  field        SFString     description ""  
  eventOut     SFTIME        bindTime  
  eventOut     SFBool        isBound  
}
```

fooBool FALSE/TRUE

fooColor [1.0 0. 0.0, 0 1 0, 0 0 1]

fooFloat [3.1415926, 12.5e-3, .0001]

fooImage <width> <height> <num components> <pixels values>

fooInt32 [17, -0xE20, -518820]

fooNode [Transform { translation 1 0 0 }
 DEF CUBE Box { }
 USE CUBE
 USE SOME_OTHER_NODE]

fooRot 0.0 1.0 0.0 3.14159265

fooString ["One, Two, Three", "He said, ¥"Immel did it!¥"]

fooTime 0.0

fooVec2f [42 666, 7 94]

fooVec3f [1 42 666, 7 94 0]

Field reference

6.7 Box

```
Box {  
  field SFVec3f size 2 2 2 # (0, ∞)  
}
```

The Box node specifies a rectangular parallelepiped box centred at (0, 0, 0) in the local coordinate system and aligned with the local coordinate axes. By default, the box measures 2 units in each dimension, from -1 to +1. The *size* field specifies the extents of the box along the X-, Y-, and Z-axes respectively and each component value shall be greater than zero. Figure 6.2 illustrates the Box node.

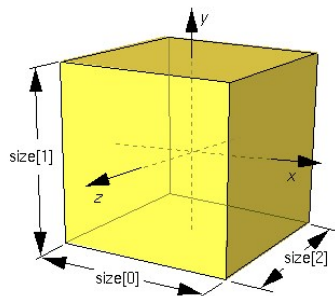


Figure 6.2 -- Box node

Textures are applied individually to each face of the box. On the front (+Z), back (-Z), right (+X) each face with the same orientation as if the image were displayed normally in 2D. On the top face the view up direction, the texture is mapped onto the face with the same orientation as if the image Y-axis toward the origin with the +Z-axis as the view up direction, the texture is mapped onto the texture coordinates of the Box.

The Box node's geometry requires outside faces only. When viewed from the inside the results



6.8 Collision

6.23 IndexedFaceSet

```
IndexedFaceSet {  
  eventIn MFInt32 set_colorIndex  
  eventIn MFInt32 set_coordIndex  
  eventIn MFInt32 set_normalIndex  
  eventIn MFInt32 set_texCoordIndex  
  exposedField SFNode color NULL  
  exposedField SFNode coord NULL  
  exposedField SFNode normal NULL  
  exposedField SFNode texCoord NULL  
  field SFBool ccw TRUE  
  field MFInt32 colorIndex [] # [-1, ∞)  
  field SFBool colorPerVertex TRUE  
  field SFBool convex TRUE  
  field MFInt32 coordIndex [] # [-1, ∞)  
  field SFFloat creaseAngle 0 # [0, ∞)  
  field MFInt32 normalIndex [] # [-1, ∞)  
  field SFBool normalPerVertex TRUE  
  field SFBool solid TRUE  
  field MFInt32 texCoordIndex [] # [-1, ∞)  
}
```

ccw: counterclockwise
convex: 凸面体

The IndexedFaceSet node represents a 3D shape formed by constructing faces (polygons) from vertices listed in the *coord* field. The *coord* field contains a Coordinate node that defines the 3D vertices referenced by the *coordIndex* field. IndexedFaceSet uses the indices in its *coordIndex* field to specify the polygonal faces by indexing into the coordinates in the Coordinate node. An index of "-1" indicates that the current face has ended and the next one begins. The last face may be (but does not have to be) followed by a "-1" index. If the greatest index in the *coordIndex* field is N, the Coordinate node shall contain N+1 coordinates (indexed as 0 to N). Each face of the IndexedFaceSet shall have:

- at least three non-coincident vertices;
- vertices that define a planar polygon;
- vertices that define a non-self-intersecting polygon.

Otherwise, The results are undefined.

The IndexedFaceSet node is specified in the local coordinate system and is affected by the transformations of its ancestors.

Descriptions of the *coord*, *normal*, and *texCoord* fields are provided in the [Coordinate](#), [Normal](#), and [TextureCoordinate](#) nodes, respectively.

Details on lighting equations and the interaction between *color* field, *normal* field, textures, materials, and geometries are provided in [4.14, Lighting model](#).

If the *color* field is not NULL, it shall contain a Color node whose colours are applied to the vertices or faces of the IndexedFaceSet as follows:

- If *colorPerVertex* is FALSE, colours are applied to each face, as follows:
 - If the *colorIndex* field is not empty, then one colour is used for each face of the IndexedFaceSet. There shall be at least as many indices in the *colorIndex* field as there are faces in the IndexedFaceSet. If the greatest index in the *colorIndex* field is N, then there shall be N+1 colours in the Color node. The *colorIndex* field shall not contain any negative entries.
 - If the *colorIndex* field is empty, then the colours in the Color node are applied to each face of the IndexedFaceSet in order. There shall be at least as many colours in the Color node as there are faces.

VRMLチュートリアル -1

- 地球が、地軸（図1のy軸）のまわりで自転しながら太陽を中心とした円軌道で公転するリアルタイム・アニメーション（Realtime Animation）をVRML言語（Virtual Reality Modeling Language, Ver.97）で記述することにより作成しなさい。但し、地球の大きさや自転・公転速度は正確でなくてもよい。
- 作成の手順を下記のように考えてみる。
- 地軸の向きは、ワールド座標系OXYZにおいて一定であり、図2のY軸をZ軸の正の向きから見て時計まわりに23.5度Z軸に関して回転した方向とする。

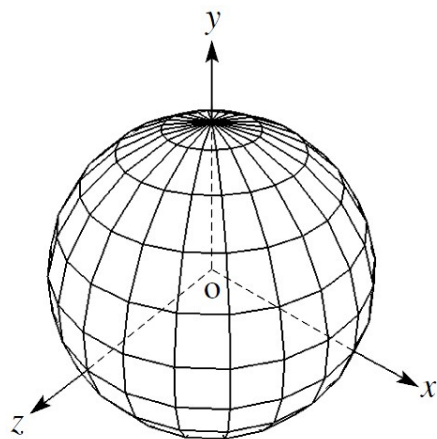


図1

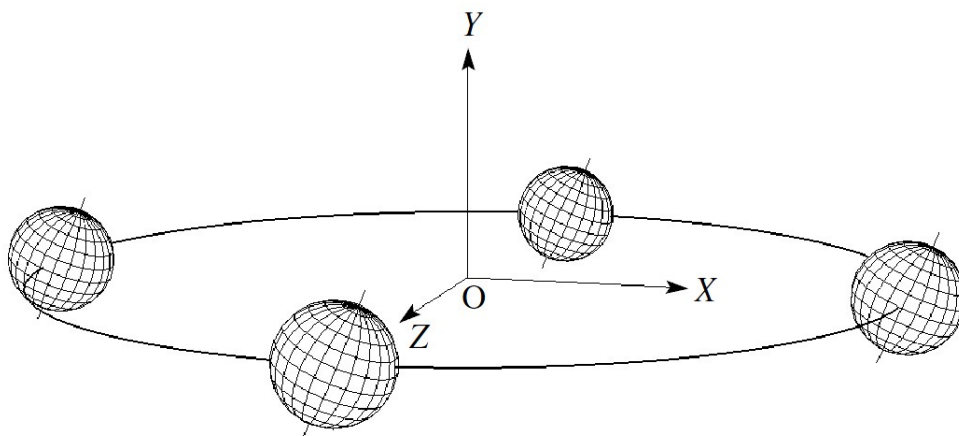


図2

チュートリアル -2

- 図1に示す地球上に固定されたローカル座標系 xyz 上の任意の点 (x, y, z) から、図2に示すワールド座標系 $OXYZ$ 上の点 (X, Y, Z) に変換する式は、同次座標表現を用いることで式①のように表せる。ただし、ワールド座標系 $OXYZ$ は、その XZ 平面が公転面と一致しているものとする。

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = A \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \dots\dots\dots ①$$

- ここで、 A は 4×4 の座標変換行列である。
- X, Y, Z 座標軸まわりの回転行列を、それぞれ $R_x(t), R_y(t), R_z(t)$ とし、平行移動行列を $M(dx, dy, dz)$ で表す。ここで、 t は回転角、 dx, dy, dz は各軸方向の平行移動成分を表すものとする。式①の A は、これらの変換行列の積となる。たとえば、以下の手順で A を求められる。
 - 最初は、ローカル座標系 xyz とワールド座標系 $OXYZ$ の位置と方向が一致しているものとする。
 - 地球の北極側（ y 軸の正の向き）から見て、地球の反時計まわりの自転を表現するために、ローカル座標値 $[x \ y \ z \ 1]^T$ に掛ける変換行列として、 Y 軸まわりの回転行列を A とする（但し、座標値右肩の T は、横ベクトルを縦ベクトルに転置させる記号）。

チュートリアル -3

3. 地軸を23.5度傾けるために、Z軸（z軸）を中心に時計まわりに23.5度回転させる行列 $R_z(-23.5\text{度})$ を2. で求めた行列Aに左側から掛ける.
4. 傾いた地球の地軸は、公転してもその方向を変えない. これを実現するために、あらかじめ、地軸を公転の回転方向とは逆方向に同期させて回転させておけば、あとで正方向に地球を公転させた時に地軸は見かけ上、方向を変えないことになる. したがって、太陽を中心に、北極側（Y軸の正の向き）から見て、地球が反時計まわりに回る公転角を θ とした時、Y軸（y軸）まわりの回転行列を3. の結果のさらに左から掛ける. その際の回転角は、 $-\theta$ となる.
5. 現在、太陽と同じ位置にある地球を、公転する半径 r の円軌道に移すために、 $M(0, 0, r)$ の行列をさらにAに左から掛ける.
6. 太陽位置を中心とした公転運動を表現するために、回転行列 $R_y(\theta)$ を、Aに左から掛ける.

VRMLを記述する準備

- テキストエディタ（TeraPad、Atom など）を準備する。
- 自身のブラウザに, VRML Viewer（Cortona3D）をインストールする。
<http://www.cortona3d.com/Products/Cortona-3D-Viewer.aspx>
- 新規ファイルを開き, 1行目に下記を入力する。これはこのファイルが, VRML（バージョン2.0=97）だという宣言を意味する。
- 2行目以降、冒頭に“#”を付けると, コメントアウトになる。

```
#VRML V2.0 utf8
```

- 半角英字のファイル名 + 拡張子「.wrl」で保存する。
- 例) test001.wrl

VRML記述

1. 背景を追加する
2. 地球を追加する
3. 地軸 (earth axis) を追加する
4. 視点を追加する
5. 地球と地軸を反時計回りに自転させる
6. 地球と地軸をZ軸に $-23.5\text{度} = -0.41015237375[\text{rad}]$ 回転させる
7. 地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる
8. 半径 r の円軌道の上に移動させる
9. 公転角 $Ry(\theta)$ で回転させる
10. 太陽を追加する

注) 各ノードを繰り返し使用したり修正する場合, 各ノードに一意である (他に同じデータがない) 名前を付けておく.

Node Reference

[6.1 Introduction](#)

[6.2 Anchor](#)

[6.3 Appearance](#)

[6.4 AudioClip](#)

[6.5 Background](#)

[6.6 Billboard](#)

[6.7 Box](#)

[6.8 Collision](#)

[6.9 Color](#)

[6.10 ColorInterpolator](#)

[6.11 Cone](#)

[6.12 Coordinate](#)

[6.13 CoordinateInterpolator](#)

[6.14 Cylinder](#)

[6.15 CylinderSensor](#)

[6.16 DirectionalLight](#)

[6.17 ElevationGrid](#)

[6.18 Extrusion](#)

[6.19 Fog](#)

[6.20 FontStyle](#)

[6.21 Group](#)

[6.22 ImageTexture](#)

[6.23 IndexedFaceSet](#)

[6.24 IndexedLineSet](#)

[6.25 Inline](#)

[6.26 LOD](#)

[6.27 Material](#)

[6.28 MovieTexture](#)

[6.29 NavigationInfo](#)

[6.30 Normal](#)

[6.31 NormalInterpolator](#)

[6.32 OrientationInterpolator](#)

[6.33 PixelTexture](#)

[6.34 PlaneSensor](#)

[6.35 PointLight](#)

[6.36 PointSet](#)

[6.37 PositionInterpolator](#)

[6.38 ProximitySensor](#)

[6.39 ScalarInterpolator](#)

[6.40 Script](#)

[6.41 Shape](#)

[6.42 Sound](#)

[6.43 Sphere](#)

[6.44 SphereSensor](#)

[6.45 SpotLight](#)

[6.46 Switch](#)

[6.47 Text](#)

[6.48 TextureCoordinate](#)

[6.49 TextureTransform](#)

[6.50 TimeSensor](#)

[6.51 TouchSensor](#)

[6.52 Transform](#)

[6.53 Viewpoint](#)

[6.54 VisibilitySensor](#)

[6.55 WorldInfo](#)

1. 背景を追加する.

#VRML V2.0 utf8

#1.背景を追加する.

```
DEF BG Background {  
  skyColor 0 0.25 0.5  
}
```

curly-brace syntax

“{” から “}” までが、
オブジェクトの宣言

2. 地球を追加する.

#VRML V2.0 utf8

#1.背景を追加する.

```
DEF BG Background {  
  skyColor 0 0.25 0.5  
}
```

#2.地球を追加する.

```
DEF EARTH_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
    texture ImageTexture {  
      url "img/earth.jpg"  
    }  
  }  
  geometry Sphere {  
    radius 3  
  }  
}
```



img



test002.wrl

“img”というフォルダ（ディレクトリ）の中に、“earth.jpg”というテクスチャ画像がある



earth.jpg



test002.wrl

同じフォルダ（ディレクトリ）に、“earth.jpg”というテクスチャ画像を配置したい場合、以下と書く。

url "earth.jpg"

地球 (Shape Node) を追加する

- 形状 (vertex, line, polygon) , 材質 (color) , テクスチャ (texture)

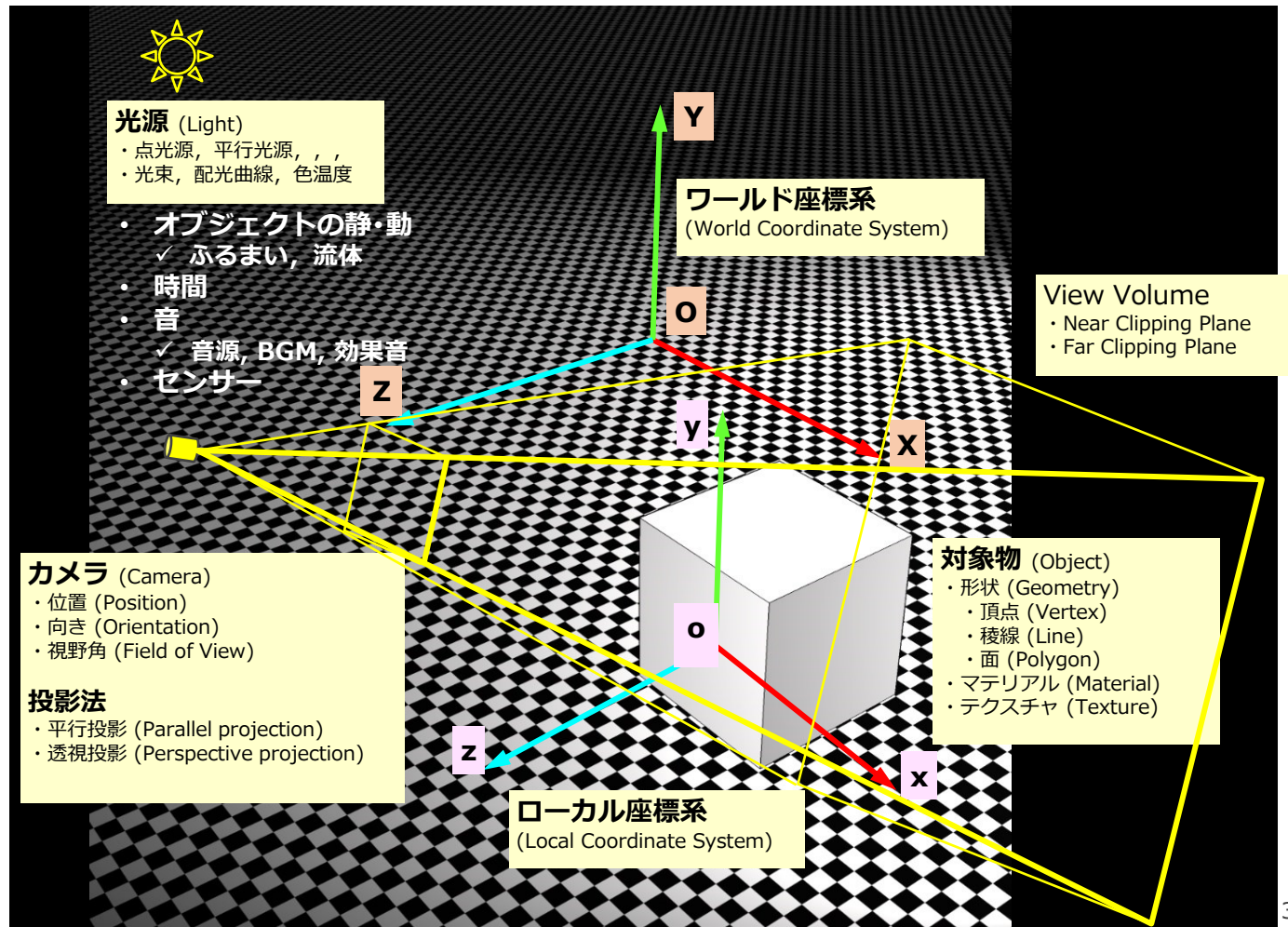
Shape

appearance Appearance

material Material

texture ImageTexture

geometry Sphere



Material

ambientIntensity

- 物体に光が当たらない面は陰や影になるが完全に真っ暗にはならず、わずかな明るさをもつ。太陽光が大気中で乱反射を繰り返し、物体の影にも散乱光が回り込むため。

diffuseColor

- 入射した光があらゆる方向に同じ強さで反射するもの。表面がざらざらした物体に見られる。

specularColor

- 物体の表面で反射により生じる光。光沢のある面では表面の一部にハイライトが生じる。

shininess

emissiveColor

- オブジェクト自身の発光色

transparency

Material

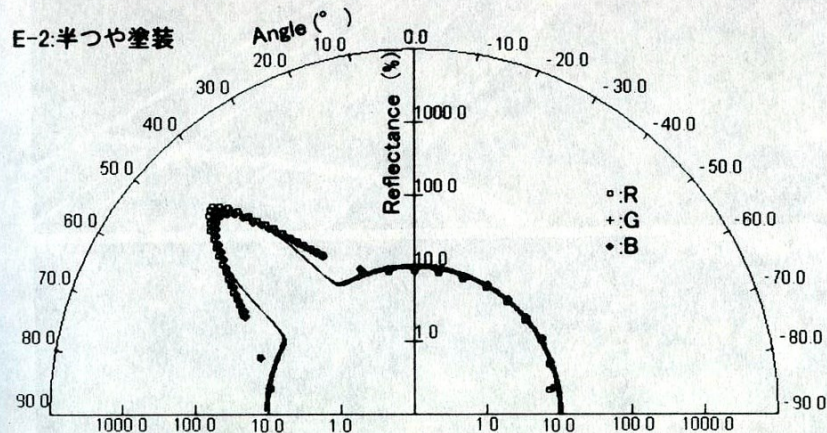
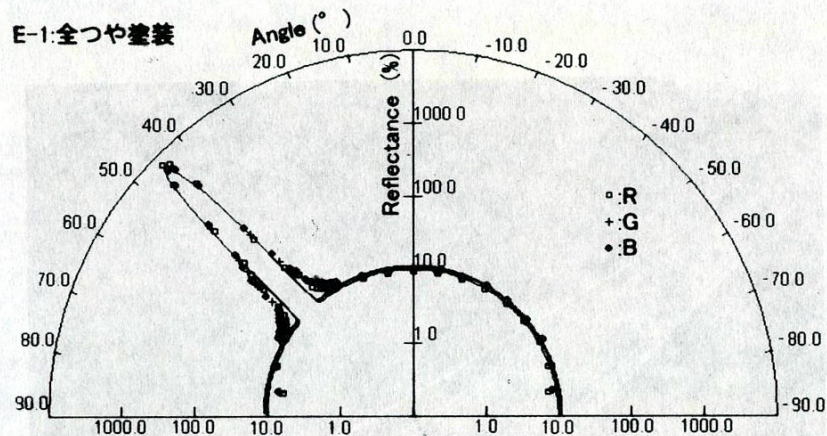


図 2: 鏡面反射を中心としたデータの解析結果

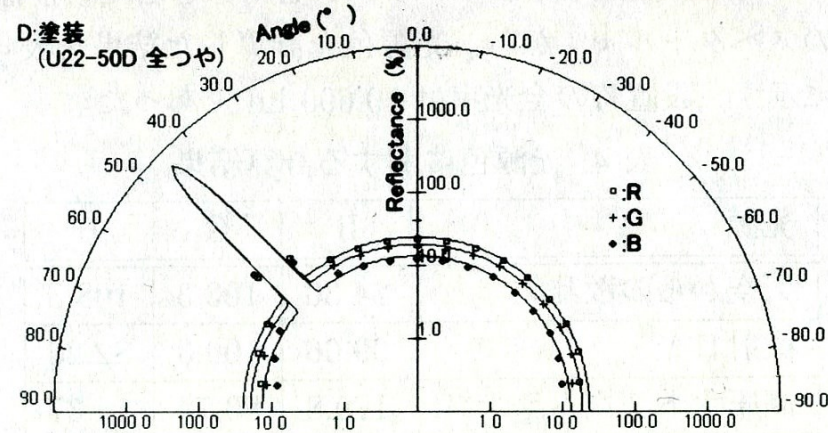
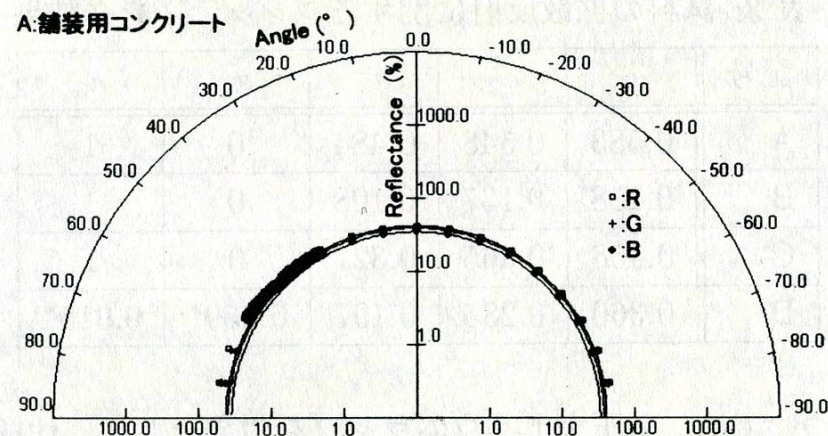


図 3: 拡散反射を中心としたデータの解析結果

ImageTexture

- 貼り付ける模様な画像 (texture/texture map) とよび, 2次元テクスチャを3次元物体面や曲面などに貼り付けて表示することをテクスチャマッピング (texture mapping) という。

3. 地軸 (earth axis) を追加する

#VRML V2.0 utf8

#1.背景を追加する.

```
DEF BG Background {  
  skyColor 0 0.25 0.5  
}
```

#2.地球を追加する.

```
DEF EARTH_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
    texture ImageTexture {  
      url "img/earth.jpg"  
    }  
  }  
  geometry Sphere {  
    radius 3  
  }  
}
```

#3.地軸を追加する.

```
DEF EARTH_AXIS_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
  }  
  geometry Cylinder {  
    radius 0.05  
    height 10  
  }  
}
```

(右列へ続く、以降も同様)

4. 視点を追加する

#VRML V2.0 utf8

#4.視点を追加する.

```
DEF VIEW Viewpoint{  
  position 0 0 80  
  description "default view"  
}
```

#1.背景を追加する.

```
DEF BG Background {  
  skyColor 0 0.25 0.5  
}
```

#2.地球を追加する.

```
DEF EARTH_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
    texture ImageTexture {  
      url "img/earth.jpg"  
    }  
  }  
  geometry Sphere {  
    radius 3  
  }  
}
```

#3.地軸を追加する.

```
DEF EARTH_AXIS_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
  }  
  geometry Cylinder {  
    radius 0.05  
    height 10  
  }  
}
```

視点 (Viewpoint Node) を追加する

Viewpoint

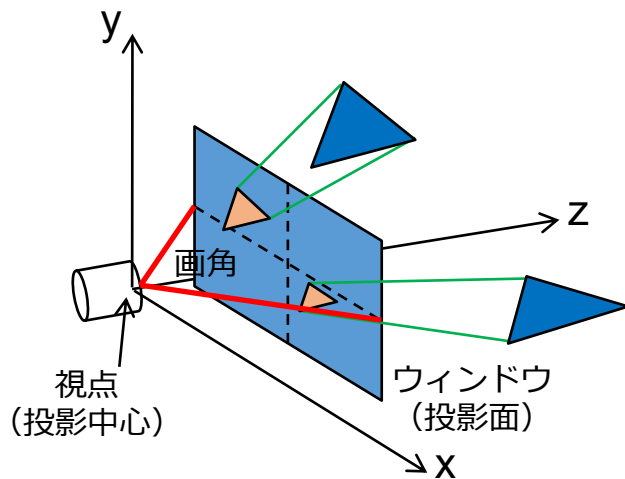
position X Y Z (Vec3f)

orientation X Y Z A (Rotation)

fieldOfView FOV (Float)

Jump TRUE/FALSE (Bool)

Description " " (String)



5. 地球と地軸を反時計回りに自転させる

#VRML V2.0 utf8

#4.視点を追加する.

```
DEF VIEW Viewpoint {  
  position 0 0 80  
  description "default view"  
}
```

#1.背景を追加する.

```
DEF BG Background {  
  skyColor 0 0.25 0.5  
}
```

#5.地球と地軸を反時計回りに自転させる.

```
DEF EARTH_ROT Transform {  
  rotation 0 1 0 0  
  children [  
    ←
```

#2.地球を追加する.

```
DEF EARTH_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
    texture ImageTexture {  
      url "img/earth.jpg"  
    }  
  }  
  geometry Sphere {  
    radius 3  
  }  
}
```

#3.地軸を追加する.

```
DEF EARTH_AXIS_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
  }  
  geometry Cylinder {  
    radius 0.05  
    height 10  
  }  
}
```

#5.地球と地軸を反時計回りに自転させる.:タイマー

```
DEF EARTH_ROT_T TimeSensor {  
  loop TRUE  
  cycleInterval 10  
}
```

#5.地球と地軸を反時計回りに自転させる:キーフレーム毎の回転角

```
DEF ROTATION_ROT OrientationInterpolator {  
  key [0, 0.25, 0.5, 0.75, 1]  
  keyValue [  
    0 1 0 0,  
    0 1 0 1.570796325,  
    0 1 0 3.14159265,  
    0 1 0 4.712388975,  
    0 1 0 6.2831853 ]  
}
```

ROUTE EARTH_ROT_T.fraction_changed TO ROTATION_ROT.set_fraction

ROUTE ROTATION_ROT.value_changed TO EARTH_ROT.rotation

自転 : Transform Node を追加する

Transform

translation X Y Z (Vec3f)

rotation X Y Z A (Rotation)

scale X Y Z (Vec3f)

scaleOrientation X Y Z A (Rotation)

center X Y Z (Vec3f)

※重要なアフィン変換ノード.

自転：キーフレーム（Key Frame）法

- 数フレームおきにオブジェクトの形や位置を指定し、その間を補間して動画を作成するアニメーション技術
- オブジェクトとは、Shape加え、Viewpoint、Lightなど。
- 要素として、物体、時間、時間に対応したキーフレーム。

6. 地球と地軸をZ軸に-23.5度 = -0.41015237375[rad]回転させる

#VRML V2.0 utf8

#4.視点を追加する.

```
DEF VIEW Viewpoint{
  position 0 0 80
  description "default view"
}
```

#1.背景を追加する.

```
DEF BG Background {
  skyColor 0 0.25 0.5
}
```

#6.地球と地軸をZ軸に-23.5度 = -0.41015237375[rad]回転させる.

```
DEF EARTH_TILT Transform {
  rotation 0 0 1 -0.41015237375
  children [
```

#5.地球と地軸を反時計回りに自転させる.

```
DEF EARTH_ROT Transform {
  rotation 0 1 0 0
  children [
```

#2.地球を追加する.

```
DEF EARTH_OBJ Shape {
  appearance Appearance {
    material Material {
      ambientIntensity 0.5
      diffuseColor 1 1 1
    }
    texture ImageTexture {
      url "img/earth.jpg"
    }
  }
  geometry Sphere {
    radius 3
  }
}
```

#3.地軸を追加する.

```
DEF EARTH_AXIS_OBJ Shape {
  appearance Appearance {
    material Material {
      ambientIntensity 0.5
      diffuseColor 1 1 1
    }
  }
  geometry Cylinder {
    radius 0.05
    height 10
  }
}
```

#5.地球と地軸を反時計回りに自転させる.:タイマー

```
DEF EARTH_ROT_T TimeSensor {
  loop TRUE
  cycleInterval 10
}
```

#5.地球と地軸を反時計回りに自転させる:キーフレーム毎の回転角

```
DEF ROTATION_ROT OrientationInterpolator {
  key [0, 0.25, 0.5, 0.75, 1]
  keyValue [
    0 1 0 0,
    0 1 0 1.570796325,
    0 1 0 3.14159265,
    0 1 0 4.712388975,
    0 1 0 6.2831853 ]
}
```

```
ROUTE EARTH_ROT_T.fraction_changed TO ROTATION_ROT.set_fraction
ROUTE ROTATION_ROT.value_changed TO EARTH_ROT.rotation
```

7. 地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる p.1

#VRML V2.0 utf8

#4.視点を追加する.

```
DEF VIEW Viewpoint {  
  position 0 0 80  
  description "default view"  
}
```

#1.背景を追加する.

```
DEF BG Background {  
  skyColor 0 0.25 0.5  
}
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる.

```
DEF EARTH_BACKSPIN Transform {  
  rotation 0 1 0 0  
  children [  
    ←
```

#6.地球と地軸をZ軸に-23.5度 = -0.41015237375[rad]回転させる.

```
DEF EARTH_TILT Transform {  
  rotation 0 0 1 -0.41015237375  
  children [  
    ←
```

#5.地球と地軸を反時計回りに自転させる.

```
DEF EARTH_ROT Transform {  
  rotation 0 1 0 0  
  children [  
    ←
```

#2.地球を追加する.

```
DEF EARTH_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
  }  
}
```

```
texture ImageTexture {  
  url "img/earth.jpg"  
}  
}  
geometry Sphere {  
  radius 3  
}  
}
```

#3.地軸を追加する.

```
DEF EARTH_AXIS_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
  }  
  geometry Cylinder {  
    radius 0.05  
    height 10  
  }  
}  
]  
}  
]  
}  
]
```


7. 地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる p.2

#5.地球と地軸を反時計回りに自転させる. :タイマー

```
DEF EARTH_ROT_T TimeSensor {  
  loop TRUE  
  cycleInterval 10  
}
```

#5.地球と地軸を反時計回りに自転させる:キーフレーム毎の回転角

```
DEF ROTATION_ROT OrientationInterpolator {  
  key [0, 0.25, 0.5, 0.75, 1]  
  keyValue [  
    0 1 0 0,  
    0 1 0 1.570796325,  
    0 1 0 3.14159265,  
    0 1 0 4.712388975,  
    0 1 0 6.2831853 ]  
}
```

```
ROUTE EARTH_ROT_T.fraction_changed TO  
ROTATION_ROT.set_fraction  
ROUTE ROTATION_ROT.value_changed TO EARTH_ROT.rotation
```

```
ROUTE EARTH_BACKSPIN_T.fraction_changed TO  
EARTH_BACKSPIN_ROTATION.set_fraction  
ROUTE EARTH_BACKSPIN_ROTATION.value_changed TO  
EARTH_BACKSPIN.rotation
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる. :タイマー

```
DEF EARTH_BACKSPIN_T TimeSensor {  
  loop TRUE  
  cycleInterval 20  
}
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる. :キーフレーム毎の回転角

```
DEF EARTH_BACKSPIN_ROTATION OrientationInterpolator {  
  key [0, 0.25, 0.5, 0.75, 1]  
  keyValue [  
    0 1 0 0,  
    0 1 0 -1.570796325,  
    0 1 0 -3.14159265,  
    0 1 0 -4.712388975,  
    0 1 0 -6.2831853 ]  
}
```



8. 半径rの円軌道上に移動させる p.1

#VRML V2.0 utf8

#4.視点を追加する.

```
DEF VIEW Viewpoint{  
  position 0 0 80  
  description "default view"  
}
```

#1.背景を追加する.

```
DEF BG Background {  
  skyColor 0 0.25 0.5  
}
```

#8.半径rの円軌道上に移動させる.

```
DEF EARTH_AXIS_TRANS Transform {  
  translation 30 0 0  
  children [
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる.

```
DEF EARTH_BACKSPIN Transform {  
  rotation 0 1 0 0  
  children [
```

#6.地球と地軸をZ軸に-23.5度 = -0.41015237375[rad]回転させる.

```
DEF EARTH_TILT Transform {  
  rotation 0 0 1 -0.41015237375  
  children [
```

#5.地球と地軸を反時計回りに自転させる.

```
DEF EARTH_ROT Transform {  
  rotation 0 1 0 0  
  children [
```

#2.地球を追加する.

```
DEF EARTH_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
  }  
}
```

```
texture ImageTexture {  
  url "img/earth.jpg"  
}  
}  
geometry Sphere {  
  radius 3  
}  
}
```

#3.地軸を追加する.

```
DEF EARTH_AXIS_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
  }  
  geometry Cylinder {  
    radius 0.05  
    height 10  
  }  
}  
]  
}  
}  
]  
}  
]  
}
```

8. 半径rの円軌道上に移動させる p.2

#5.地球と地軸を反時計回りに自転させる. :タイマー

```
DEF EARTH_ROT_T TimeSensor {  
  loop TRUE  
  cycleInterval 10  
}
```

#5.地球と地軸を反時計回りに自転させる:キーフレーム毎の回転角

```
DEF ROTATION_ROT OrientationInterpolator {  
  key [0, 0.25, 0.5, 0.75, 1]  
  keyValue [  
    0 1 0 0,  
    0 1 0 1.570796325,  
    0 1 0 3.14159265,  
    0 1 0 4.712388975,  
    0 1 0 6.2831853 ]  
}
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる. :タイマー

```
DEF EARTH_BACKSPIN_T TimeSensor {  
  loop TRUE  
  cycleInterval 20  
}
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる. :キーフレーム毎の回転角

```
DEF EARTH_BACKSPIN_ROTATION OrientationInterpolator {  
  key [0, 0.25, 0.5, 0.75, 1]  
  keyValue [  
    0 1 0 0,  
    0 1 0 -1.570796325,  
    0 1 0 -3.14159265,  
    0 1 0 -4.712388975,  
    0 1 0 -6.2831853 ]  
}
```

```
ROUTE EARTH_ROT_T.fraction_changed TO  
ROTATION_ROT.set_fraction  
ROUTE ROTATION_ROT.value_changed TO EARTH_ROT.rotation
```

```
ROUTE EARTH_BACKSPIN_T.fraction_changed TO  
EARTH_BACKSPIN_ROTATION.set_fraction  
ROUTE EARTH_BACKSPIN_ROTATION.value_changed TO  
EARTH_BACKSPIN.rotation
```



9. 公転角 $R_y(\theta)$ で回転させる p.1

#VRML V2.0 utf8

#4.視点を追加する.

```
DEF VIEW Viewpoint{  
  position 0 0 80  
  description "default view"  
}
```

#1.背景を追加する.

```
DEF BG Background {  
  skyColor 0 0.25 0.5  
}
```

#9.公転角 $R_y(\theta)$ で回転させる.

```
DEF EARTH Transform {  
  rotation 0 1 0 0  
  children [  

```

#8.半径 r の円軌道上に移動させる.

```
DEF EARTH_AXIS_TRANS Transform {  
  translation 30 0 0  
  children [  

```

#7.地球と地軸を公転角 θ と反対角となる $R_y(-\theta)$ で回転させる.

```
DEF EARTH_BACKSPIN Transform {  
  rotation 0 1 0 0  
  children [  

```

#6.地球と地軸をZ軸に -23.5 度 $=-0.41015237375$ [rad]回転させる.

```
DEF EARTH_TILT Transform {  
  rotation 0 0 1 -0.41015237375  
  children [  

```

#5.地球と地軸を反時計回りに自転させる.

```
DEF EARTH_ROT Transform {  
  rotation 0 1 0 0  
  children [  

```

#2.地球を追加する.

```
DEF EARTH_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
    texture ImageTexture {  
      url "img/earth.jpg"  
    }  
  }  
  geometry Sphere {  
    radius 3  
  }  
}
```

#3.地軸を追加する.

```
DEF EARTH_AXIS_OBJ Shape {  
  appearance Appearance {  
    material Material {  
      ambientIntensity 0.5  
      diffuseColor 1 1 1  
    }  
  }  
  geometry Cylinder {  
    radius 0.05  
    height 10  
  }  
}  
]  
}  
]  
}  
]  
}  
]  
}
```

9. 公転角 $Ry(\theta)$ で回転させる p.2

#5.地球と地軸を反時計回りに自転させる. :タイマー

```
DEF EARTH_ROT_T TimeSensor {  
  loop TRUE  
  cycleInterval 10  
}
```

#5.地球と地軸を反時計回りに自転させる:キーフレーム毎の回転角

```
DEF ROTATION_ROT OrientationInterpolator {  
  key [0, 0.25, 0.5, 0.75, 1]  
  keyValue [  
    0 1 0 0,  
    0 1 0 1.570796325,  
    0 1 0 3.14159265,  
    0 1 0 4.712388975,  
    0 1 0 6.2831853 ]  
}
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる. :タイマー

```
DEF EARTH_BACKSPIN_T TimeSensor {  
  loop TRUE  
  cycleInterval 20  
}
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる. :キーフレーム毎の回転角

```
DEF EARTH_BACKSPIN_ROTATION OrientationInterpolator {  
  key [0, 0.25, 0.5, 0.75, 1]  
  keyValue [  
    0 1 0 0,  
    0 1 0 -1.570796325,  
    0 1 0 -3.14159265,  
    0 1 0 -4.712388975,  
    0 1 0 -6.2831853 ]  
}
```

#9.公転角 $Ry(\theta)$ で回転させる. :タイマー

```
DEF EARTH_T TimeSensor {  
  loop TRUE  
  cycleInterval 20  
}
```

#9.公転角 $Ry(\theta)$ で回転させる. :キーフレーム毎の回転角

```
DEF ROTATION OrientationInterpolator {  
  key [0, 0.25, 0.5, 0.75, 1]  
  keyValue [  
    0 1 0 0,  
    0 1 0 1.570796325,  
    0 1 0 3.14159265,  
    0 1 0 4.712388975,  
    0 1 0 6.2831853 ]  
}
```

```
ROUTE EARTH_ROT_T.fraction_changed TO  
ROTATION_ROT.set_fraction  
ROUTE ROTATION_ROT.value_changed TO EARTH_ROT.rotation
```

```
ROUTE EARTH_BACKSPIN_T.fraction_changed TO  
EARTH_BACKSPIN_ROTATION.set_fraction  
ROUTE EARTH_BACKSPIN_ROTATION.value_changed TO  
EARTH_BACKSPIN.rotation
```

```
ROUTE EARTH_T.fraction_changed TO ROTATION.set_fraction  
ROUTE ROTATION.value_changed TO EARTH.rotation
```



10. 太陽を追加する p.1

#VRML V2.0 utf8

#4.視点を追加する.

```
DEF VIEW Viewpoint{  
  position 0 0 80  
  description "default view"  
}
```

#1.背景を追加する.

```
DEF BG Background {  
  skyColor 0 0.25 0.5  
}
```

#9.公転角 $R_y(\theta)$ で回転させる.

```
DEF EARTH Transform {  
  rotation 0 1 0 0  
  children [
```

#8.半径 r の円軌道の上に移動させる.

```
  DEF EARTH_AXIS_TRANS Transform {  
    translation 30 0 0  
    children [
```

#7.地球と地軸を公転角 θ と反対角となる $R_y(-\theta)$ で回転させる.

```
    DEF EARTH_BACKSPIN Transform {  
      rotation 0 1 0 0  
      children [
```

#6.地球と地軸をZ軸に-23.5度 = -0.41015237375[rad]回転させる.

```
      DEF EARTH_TILT Transform {  
        rotation 0 0 1 -0.41015237375  
        children [
```

#5.地球と地軸を反時計回りに自転させる.

```
        DEF EARTH_ROT Transform {  
          rotation 0 1 0 0  
          children [
```

#2.地球を追加する.

```
        DEF EARTH_OBJ Shape {  
          appearance Appearance {  
            material Material {  
              ambientIntensity 0.5  
              diffuseColor 1 1 1  
            }  
            texture ImageTexture {  
              url "img/earth.jpg"  
            }  
          }  
          geometry Sphere {  
            radius 3  
          }  
        }  
      ]  
    }  
  ]  
}
```

#3.地軸を追加する.

```
    DEF EARTH_AXIS_OBJ Shape {  
      appearance Appearance {  
        material Material {  
          ambientIntensity 0.5  
          diffuseColor 1 1 1  
        }  
      }  
      geometry Cylinder {  
        radius 0.05  
        height 10  
      }  
    }  
  ]  
}
```

10. 太陽を追加する p.2

#10.太陽を追加する.

```
DEF SUN_OBJ Shape {
  appearance Appearance {
    material Material {
      ambientIntensity 1
      emissiveColor 1 1 0
    }
  }
  geometry Sphere {
    radius 10
  }
}
```

#5.地球と地軸を反時計回りに自転させる. :タイマー

```
DEF EARTH_ROT_T TimeSensor {
  loop TRUE
  cycleInterval 20
}
```

#5.地球と地軸を反時計回りに自転させる:キーフレーム毎の回転角

```
DEF ROTATION_ROT OrientationInterpolator {
  key [0, 0.25, 0.5, 0.75, 1]
  keyValue [
    0 1 0 0,
    0 1 0 1.570796325,
    0 1 0 3.14159265,
    0 1 0 4.712388975,
    0 1 0 6.2831853 ]
}
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる. :タイマー

```
DEF EARTH_BACKSPIN_T TimeSensor {
  loop TRUE
  cycleInterval 20
}
```

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる. :キーフレーム毎の回転角

```
DEF EARTH_BACKSPIN_ROTATION OrientationInterpolator {
  key [0, 0.25, 0.5, 0.75, 1]
  keyValue [
    0 1 0 0,
    0 1 0 -1.570796325,
    0 1 0 -3.14159265,
    0 1 0 -4.712388975,
    0 1 0 -6.2831853 ]
}
```

#9.公転角 $Ry(\theta)$ で回転させる. :タイマー

```
DEF EARTH_T TimeSensor {
  loop TRUE
  cycleInterval 20
}
```

#9.公転角 $Ry(\theta)$ で回転させる. :キーフレーム毎の回転角

```
DEF ROTATION OrientationInterpolator {
  key [0, 0.25, 0.5, 0.75, 1]
  keyValue [
    0 1 0 0,
    0 1 0 1.570796325,
    0 1 0 3.14159265,
    0 1 0 4.712388975,
    0 1 0 6.2831853 ]
}
```

```
ROUTE EARTH_ROT_T.fraction_changed TO
ROTATION_ROT.set_fraction
ROUTE ROTATION_ROT.value_changed TO EARTH_ROT.rotation
```

```
ROUTE EARTH_BACKSPIN_T.fraction_changed TO
EARTH_BACKSPIN_ROTATION.set_fraction
ROUTE EARTH_BACKSPIN_ROTATION.value_changed TO
EARTH_BACKSPIN.rotation
```

```
ROUTE EARTH_T.fraction_changed TO ROTATION.set_fraction
ROUTE ROTATION.value_changed TO EARTH.rotation
```


VRML Tutorial test010.wrl 構造

#4.視点を追加する. VIEW

#1.背景を追加する. BG

#9.公転角 $Ry(\theta)$ で回転させる. EARTH

#8.半径 r の円軌道上に移動させる. EARTH_AXIS_TRANS

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる. EARTH_BACKSPIN

#6.地球と地軸をZ軸に -23.5 度 $= -0.41[\text{rad}]$ 回転させる. EARTH_TILT

#5.地球と地軸を反時計回りに自転させる. EARTH_ROT

#2.地球を追加する. EARTH_OBJ

#3.地軸を追加する. EARTH_AXIS_OBJ

#10.太陽を追加する. SUN_OBJ

#5.地球と地軸を反時計回りに自転させる:タイマー EARTH_ROT_T

#5.地球と地軸を反時計回りに自転させる:キーフレーム毎の回転角 ROTATION_ROT

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる:タイマー EARTH_BACKSPIN_T

#7.地球と地軸を公転角 θ と反対角となる $Ry(-\theta)$ で回転させる:キーフレーム毎の回転角

EARTH_BACKSPIN_ROTATION

#9.公転角 $Ry(\theta)$ で回転させる:タイマー EARTH_T

#9.公転角 $Ry(\theta)$ で回転させる:キーフレーム毎の回転角 ROTATION

test010.wrlを拡張してみよう

- 太陽の光で、地球を照らしてみよう。
- 地球の周りに、月を自転、公転させてみよう。
- マウスでクリック、ドラッグして、インタラクティブな操作を加えてみよう。

などなど！

練習問題 -2 (15分)

以下の文章は、景観シミュレーションに関して述べたものである。

(a) ~ (i) に適当な数値、語句を入れなさい。

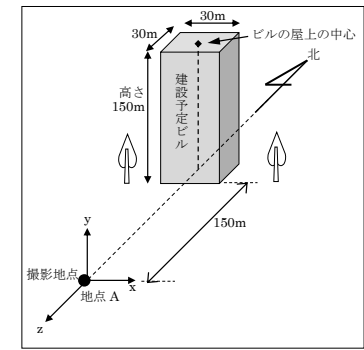


図1

都心部で計画中のビル（幅30m，奥行き30m，高さ150m）をフォトモンタージュ法により景観シミュレーションを行うことになった。これは、ビル建設以前に現状の景観をデジタル写真で撮影し、その画像に建設予定のビルの3DCG画像を合成し、ビル建設後の景観を予測する手法である。ここで、3DCGは短縮形の名称であり、フルスペリングを英語で記述すると、(a) _____ となる。

ビル建設予定地や撮影場所一帯の地面の状態は水平である。現在の景観の画像は、図のようにビル建設予定地から南側に150m離れた地点A（地面上）から撮影する。

撮影では、ビルの上端から下端までが画像の垂直方向の上端と下端に丁度入るようにしたい。そこで、ズームレンズを操作して、垂直方向の視野角（垂直画角）を整数値で (b) _____ 度に設定して撮影した。この場合、画像のアスペクト比を水平画角：垂直画角 = 16:9 とすると、水平方向の視野角（水平画角）は整数値で (c) _____ 度となる。

次に、ビルの3DCG画像を作成する。まず、(d) _____ 座標系を用いてビルの形状モデルを作成した後、別途作成した添景モデル（車や人など）などと共に、実際の位置関係に合わせて、(e) _____ 座標系に配置する。ここでは、撮影地点を原点、長さの単位をメートル（m）とし、水平東向きにx軸の正方向、鉛直上向きにy軸の正方向、水平南向きにz軸の正方向となる (e) _____ 座標系を考えることにした。

合成用のビルの3DCG画像を作成するためには、(f) _____ 投影を用い、(e) _____ 座標系において、視点の座標を ((g) _____)，((h) _____)，((i) _____)，ビルの屋上の中心座標を ((j) _____)，((k) _____)，((l) _____) とすればよい。

また、撮影時には、太陽が丁度真南で45°の角度に位置していた。3DCGではこの太陽光を平行光源で定義した場合、光源を表す単位ベクトルは、(e) _____ 座標系において、((m) _____)，((n) _____)，((o) _____) とすればよい。

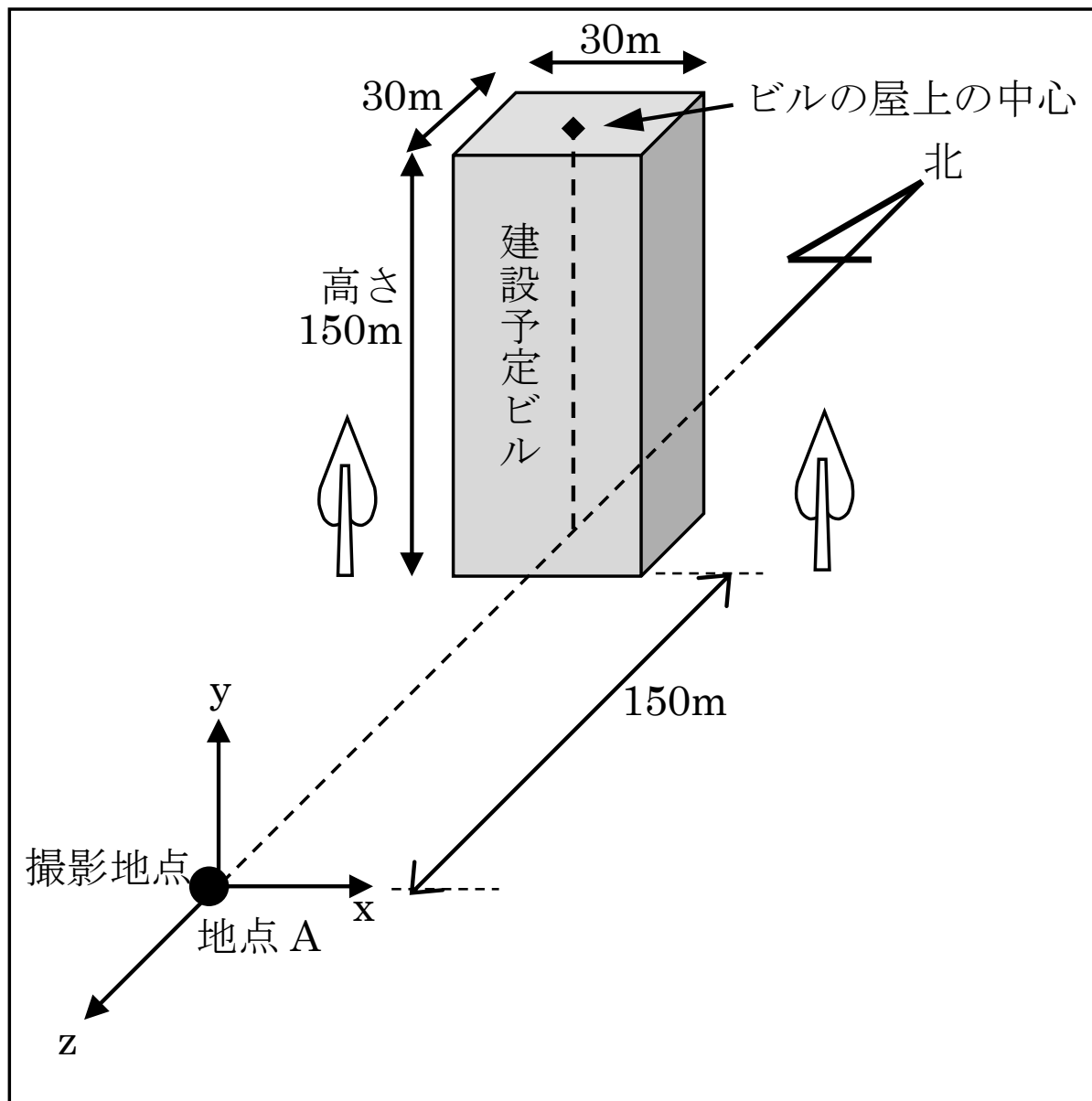


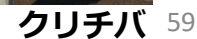
図1

国際感覚

Think globally, Act locally

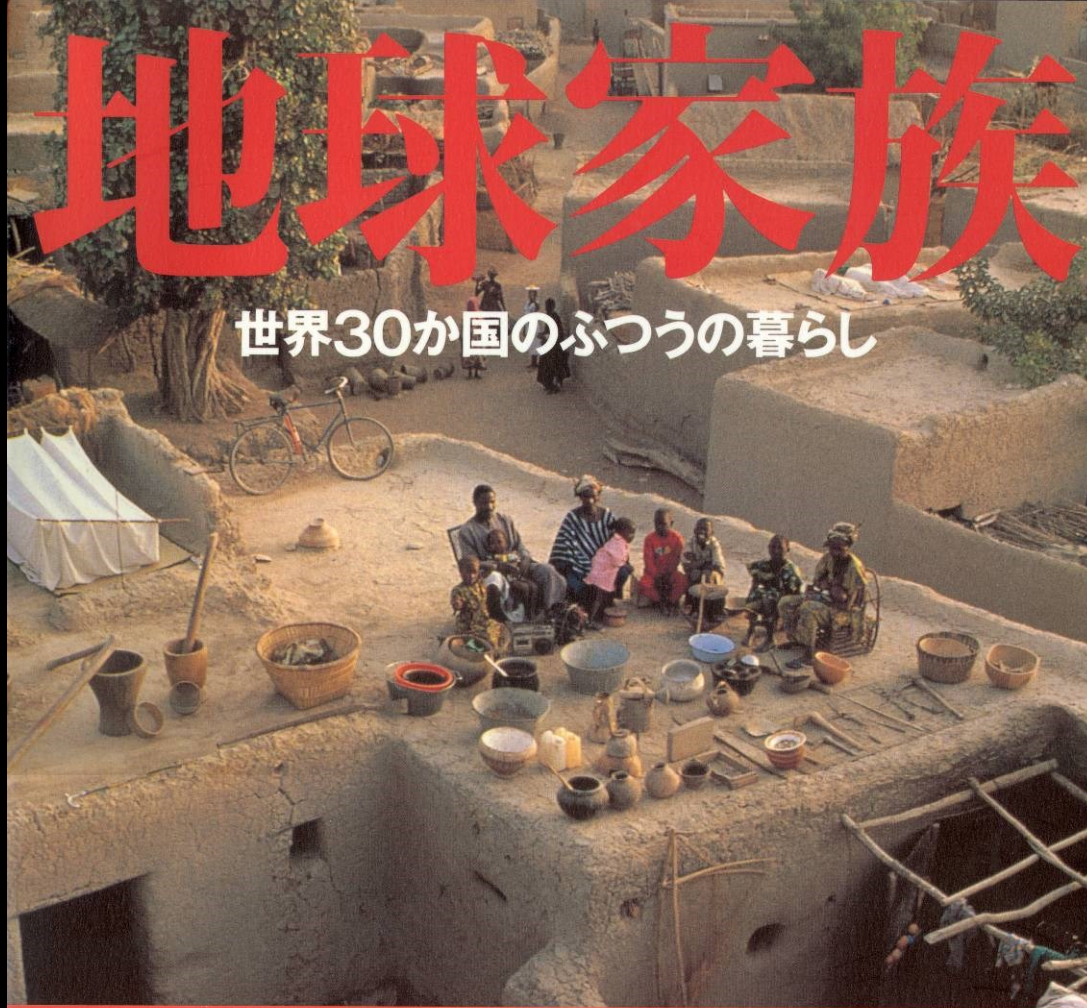
「地球規模で考え、足元から行動せよ」

国内外の町を訪問したら、ジモティに出会って話をしよう。日本との違いが見えてくる。



地球家族

世界30か国のふつうの暮らし



「申し訳ありませんが、家の中の物を全部、
家の前に出して写真を撮らせて下さい。」

by ピーター・メンツェル

国際家族年の地球を一周、国連(開発基金、人口基金)、世界銀行後援、
前人未到の大プロジェクト。 TOTO出版

学会

CAADRIA 2013 を例として

Computer Aided Architectural Design Research In Asia

Call for Abstracts	15 Jun 2012
Abstract Submission	09 Sep 2012
Abstract Acceptance Notice	01 Oct 2012
Full Paper Submission	02 Dec 2012
Full Paper Acceptance Notice	15 Jan 2013
Registration Deadline	15 Feb 2013
Camera-ready Paper Submission	17 Feb 2013
Conference	15-18 May 2013



VRML97 (Monterey, CA)



4町パティオ（高松, 2005-07）

道路用地にパラソルを常設。沿道の土地建物所有者、テナントで運営管理。ハートビートプラン・泉英明 氏とコラボ。検討過程では、VRを使いながらパティオの将来像を何度も議論した。



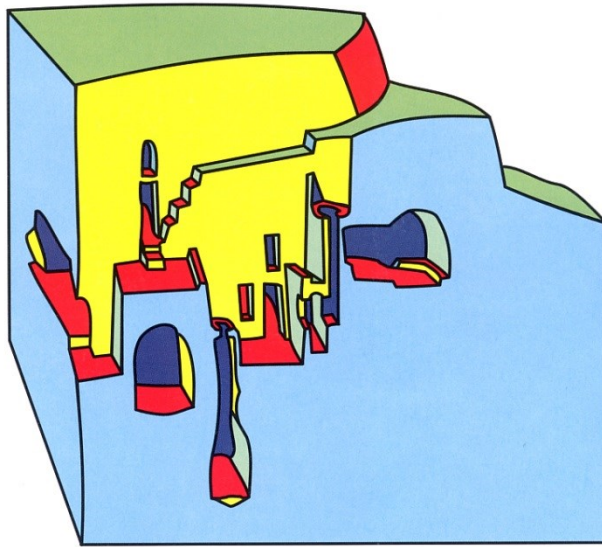
LERNEN VON SANTORIN

DIE ÖKOLOGIE DES LEBENSRA

ÖKOLOGIE

NATUR

KULTUR



EFTHYMIOS WARLAMIS

ΝΙΚΟΣ ΡΗΓΟΠΟΥΛΟΣ
NIKOS RIGOPOULOS

ΟΙΑ ΦΩΣ ΚΑΙ ΧΡΩΜΑ ΤΗΣ ΣΑΝΤΟΡΙΝΗΣ LIGHT AND COLOUR OF SANTORINI

ΤΟΡΙΝΗΣ
ANTORINI



Ο Νίκος Ρηγόπουλος γεννήθηκε στην Αθήνα το 1971. Από το 1992 ασχολείται με την φωτογραφία. Μέχρι το 2000 διατηρούσε φωτογραφικό εργαστήριο στην Αθήνα και κατά την ίδια περίοδο ταξίδεψε στην Ελλάδα και την Ευρώπη φωτογραφίζοντας.

Έχει κάνει πολλές ατομικές και ομαδικές εκθέσεις στην

小舟木工コ村（近江八幡, 2004-08）

持続可能な社会をつくるためにはどのように行動すべきか考えるべきフィールドづくりを目指す。住民自らがこれまでの暮らしを見直し、コミュニティレベルで、太陽光エネルギーの利用、雨水利用、生ごみの堆肥化などに取組むことで、人と環境が共生する新しいライフスタイルを目指す。



地域の畑と食卓をつなぐ



自然とつながる住まい



緑あふれるまちなみ作り



大学やNPOとの連携

Next Gene 21+ (台湾, 2007-08)

台湾内外20数名ものの建築家によるハウジング計画。日本からの建築家として、安藤忠雄、隈研吾、平田晃久が参加。我々のチームは、台北郊外での敷地での新たなライフスタイルを描いた。ベネチアビエンナーレ2008出展。

