大規模並列可視化

小野 謙二

理化学研究所 計算科学研究機構
東京大学 生産技術研究所
神戸大学 システム情報学研究科

TOC

- 可視化のアウトライン
  - イメージ生成
  - 可視化技術
- 大規模並列計算
- 並列レンダリング
- 並列分散ファイル管理
- HIVEシステム
- Hands-on

可視化のアウトライン

可視化

- Scientific Visualization
  - 科学技術データを対象とした視覚化
- Information Visualization
  - 情報の視覚化
  - データマイニング
- 知的可視化
  - Visual Analysis
    - 視覚的な効果を利用した分析
  - 知識情報処理と可視化の連携
- 適用分野
  - 医学，生物学，天文学，地球科学，気象学，機械工学，．．

可視化の目的

- 現象の把握と理解
  - 分析と発見
- エンジニアリング
  - 何が原因で性能がでないのか？
  - 改善はどうすればよいか？
- プレゼンテーション
  - 論文発表や予算獲得
    - 説得力のある画像･動画

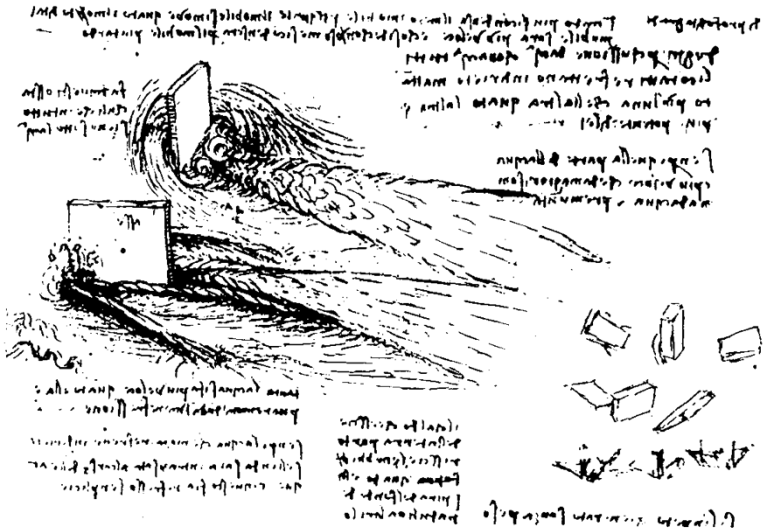
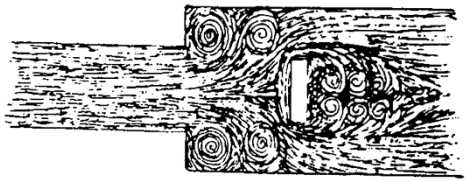


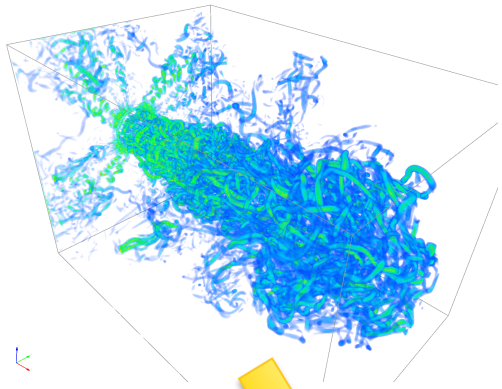
図 1.14　障害物のまわりの流れ(レオナルド・ダ・ビンチのノートから)



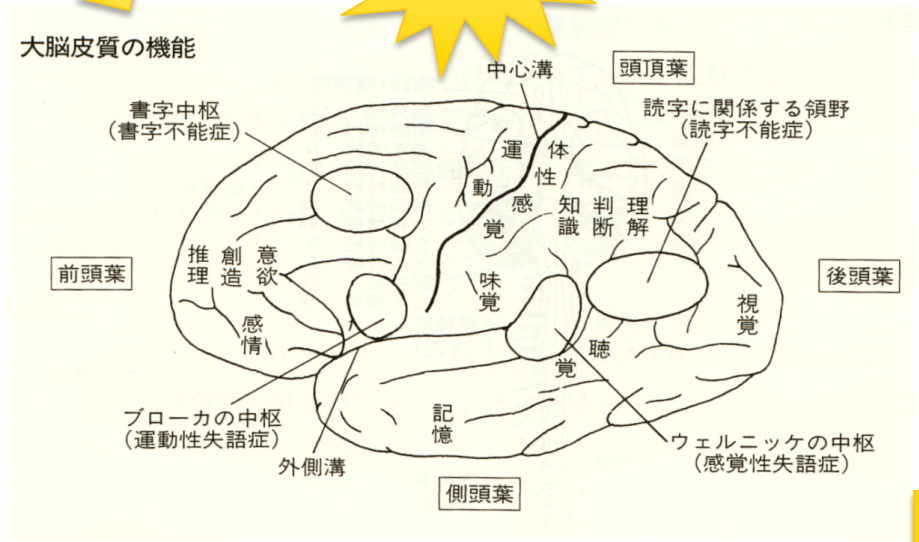
a)　広がり流れの渦と物体背後の渦

b)　流線形

イメージの重要性



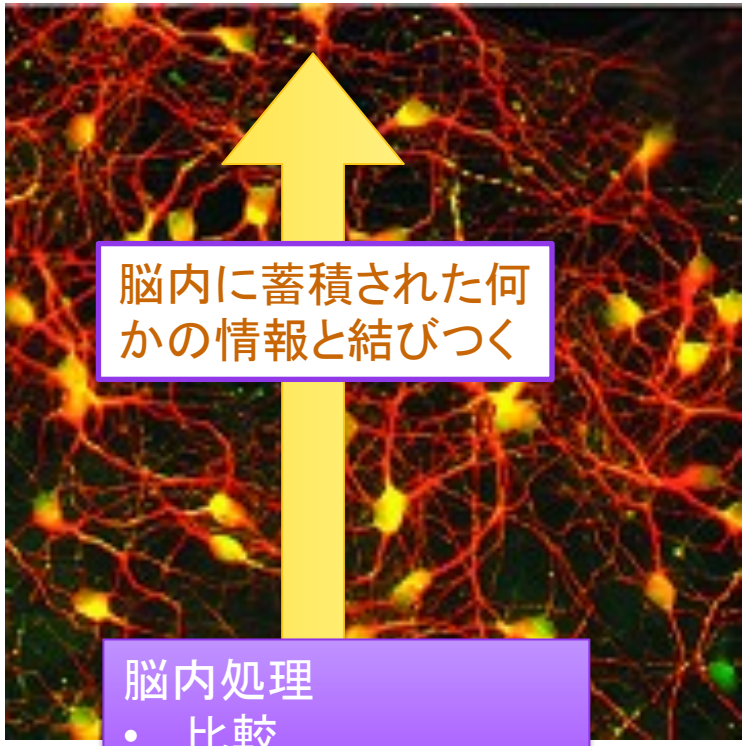
刺激



脳内マッピング

アイデア！

論理的考察へ



脳内に蓄積された何かの情報と結びつく

脳内処理
- 比較
- 類似検索
- パターンマッチング

イメージ生成

レンダリング

イメージを作る基本的なしくみ

- データを読み込む
  - シミュレーションのデータを取り込む
  - 様々なデータ型とフォーマットがある

- イメージをつくり、ディスプレイに表示する
  - コンピュータグラフィクス技術の応用
  - ピクセル（二次元）空間に三次元の世界を写し取る
  - 眼の網膜とほぼ同じしくみを実装
    - 光源からの光が網膜に届く過程を模擬
    - 光源、視点、スクリーン、三次元の物体
    - レイトレーシング技法

データ型

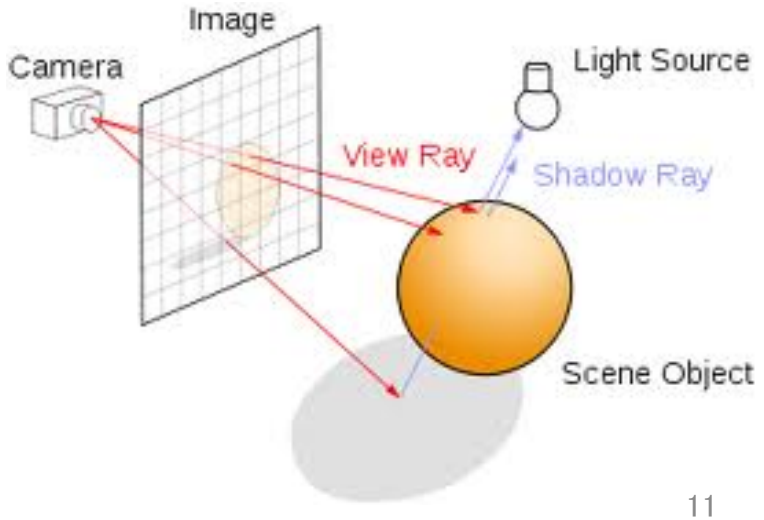
- スカラデータ
- ベクトルデータ
- テンソルデータ
- 時系列データ
- 粒子データ

データ構造とファイルフォーマット

- 構造格子
  - Cartesian
  - Octree
  - 一般曲線座標系
- 非構造格子
- 点群

レイトレーシング法

- 網膜の代わりにスクリーンを用意
- 光源からの光線をたどりピクセルの色（RGB）を決める
- 交差判定により，反射と拡散を計算する
- 物体の材質により，色を吸収することも考慮できる
- 高品質のためには沢山のレイを使う

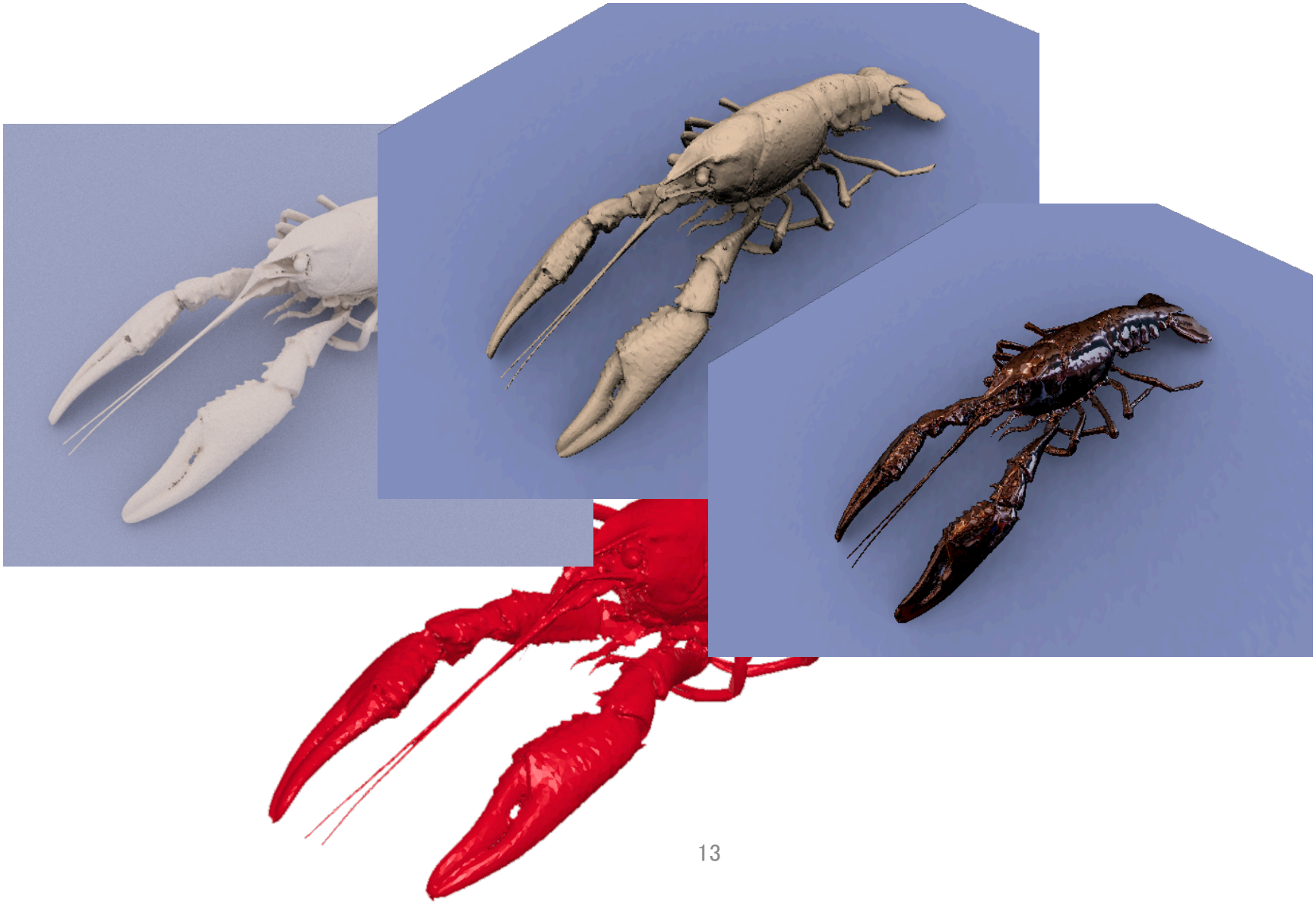


Kilauea　レイトレーサー

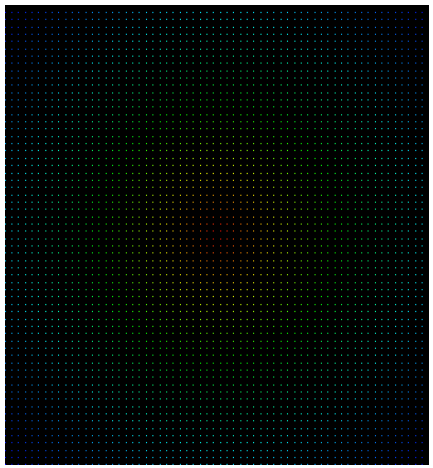
787,255 triangles, 1 area light, 1280 x 692
18 machines : 9min 10sec

Toshi Kato Square USA, http://www.squareusa.com/kilauea/

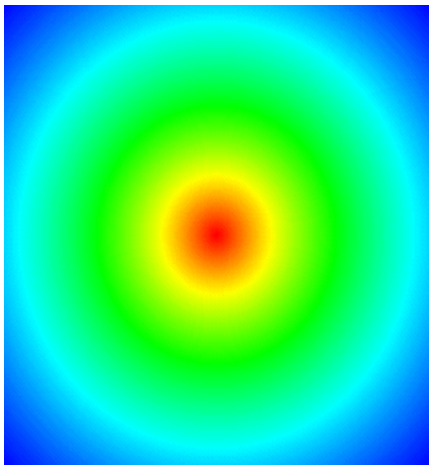
Photo Realistic Visual Effects



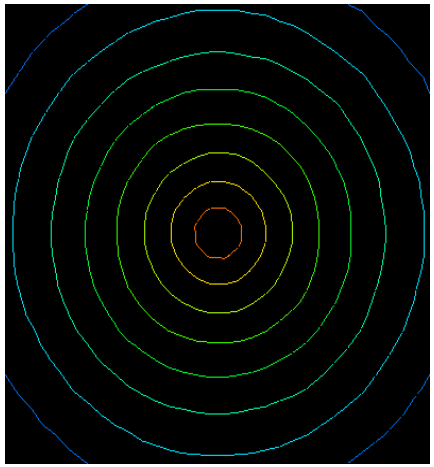
スカラーデータの可視化法



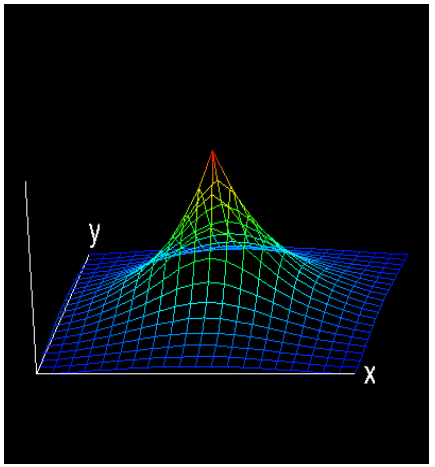
点描



フリンジ

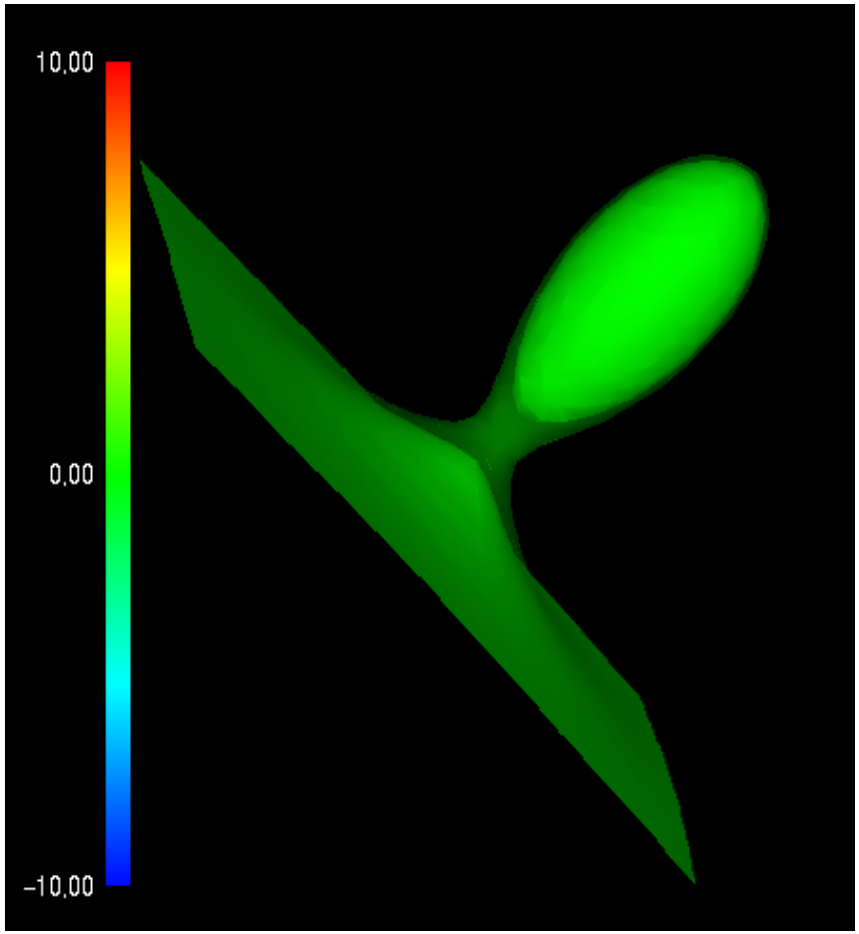


等高線

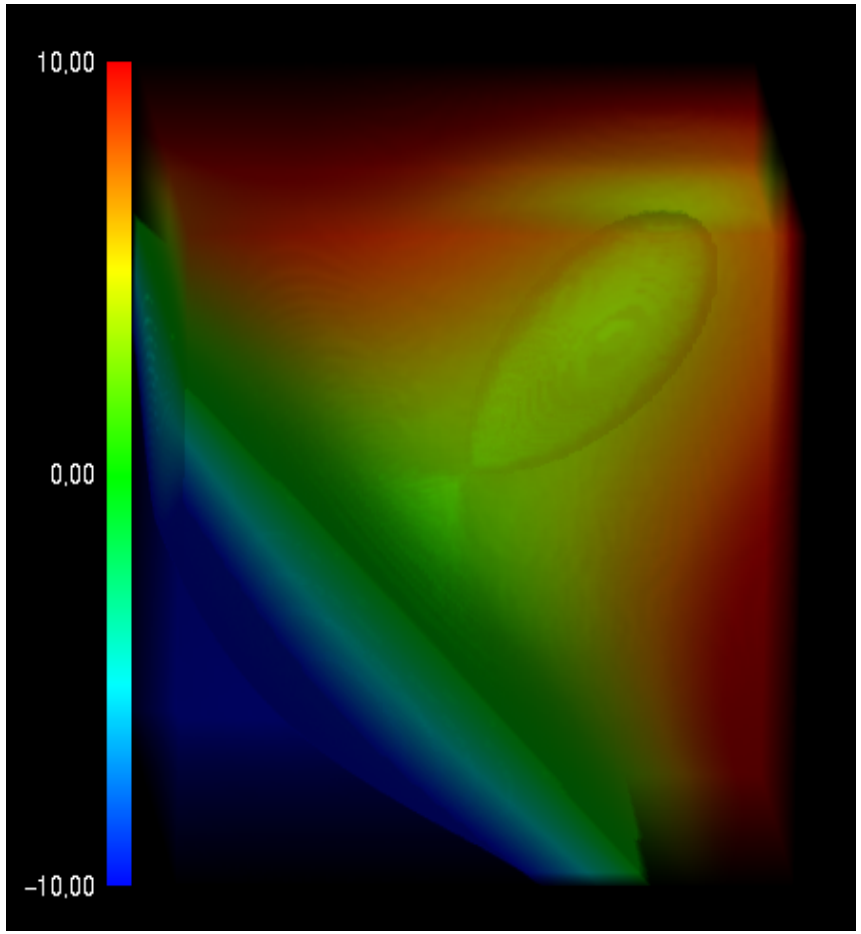


等値面

等値面による空間分布の把握



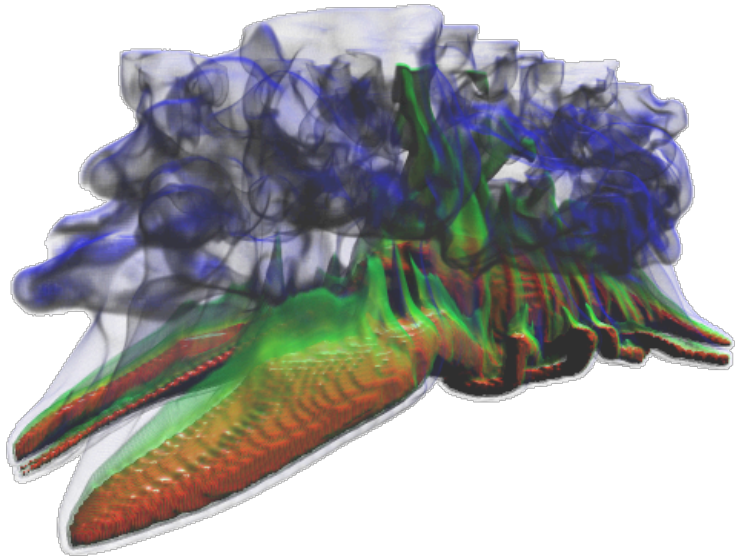
単一の等値面



複数等値面+半透明表示

Volume Rendering

- ボリュームデータのトランスルーセントな表現
  - 光線の追跡を行い、光の減衰などを考慮して画像を生成
  - 内部構造の透過性を有し，全体像の把握に優れる
  - 複雑な組織や構造・挙動を理解する
- 人の持つ優れた深度識別能力やパターン認識能力を利用
- 伝達関数の設計が重要
  - 試行錯誤，支援環境

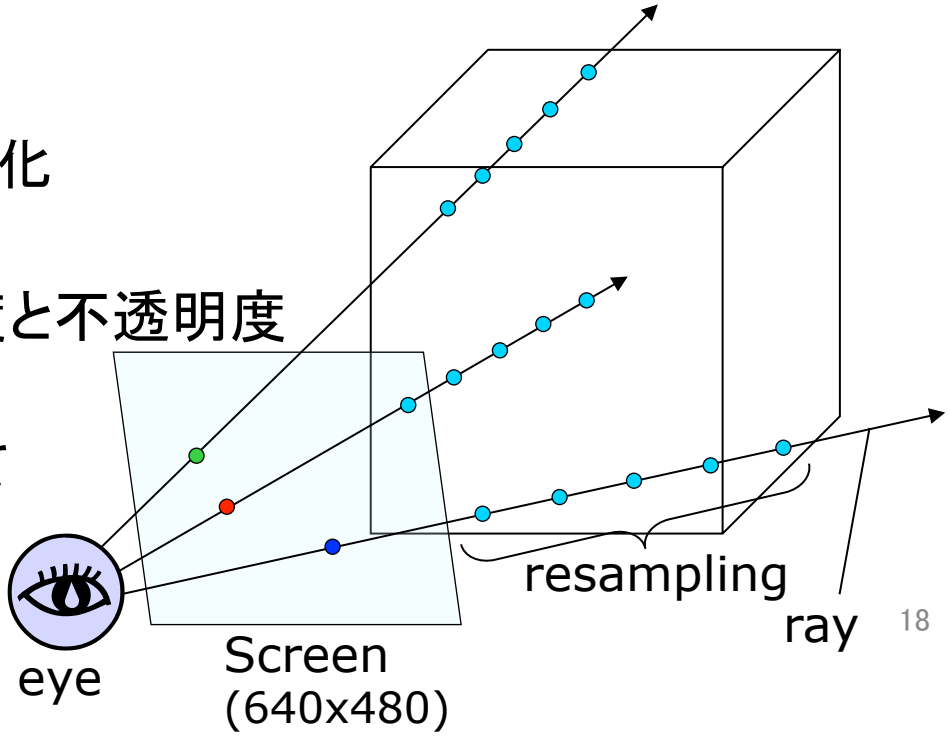


ボリュームレンダリング

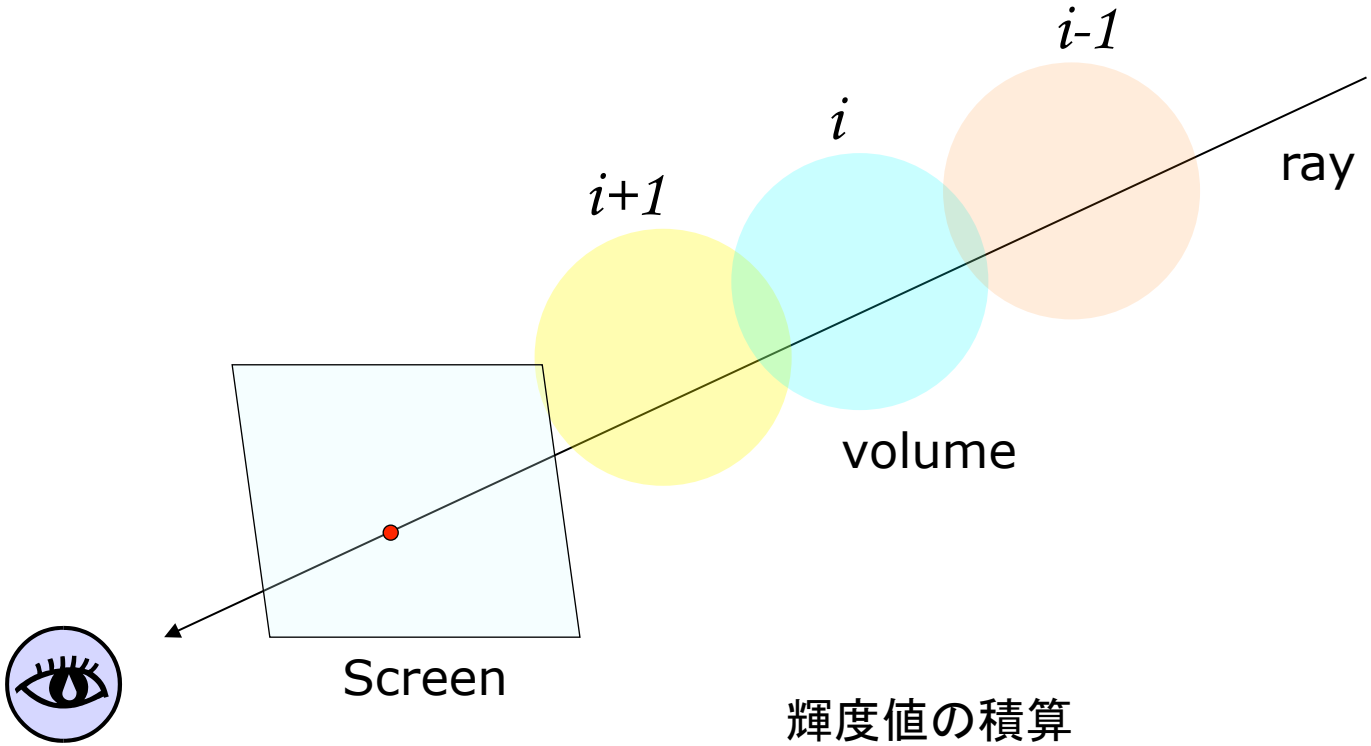
- レイトレーシングに透明度を導入
- 外形と内部を同時に描画
- パラメータ設定にコツ

Ray Casting　Levoy ’88

- ボリュームを半透明のゲル状物質と考える
  - 光線が通過するときの輝度変化を表す光学モデルを使用
- スクリーンの配置
  - ビューイング設定
- リサンプリング
  - 視線ベクトル方向の標本化
- 伝達関数による変換
  - フィールド値→カラー輝度と不透明度
- 画素の値の決定
  - リサンプリング点に対して
  - カラー輝度と不透明度の
  - 積和をとる



透過率



輝度値の積算

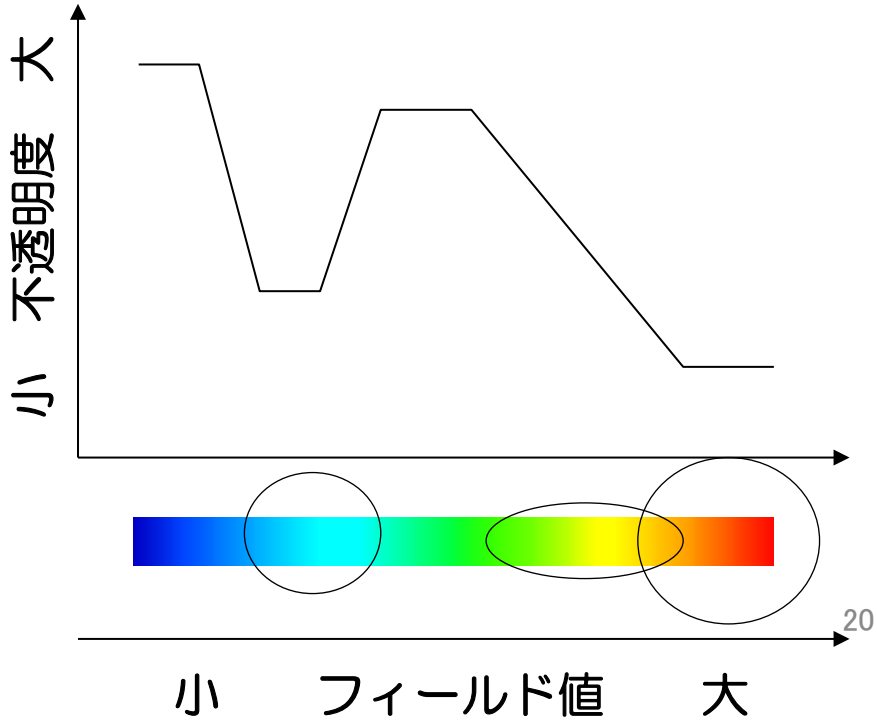
$I_{out} = (1-\alpha)I_{in}+I_i\alpha$

I 輝度値

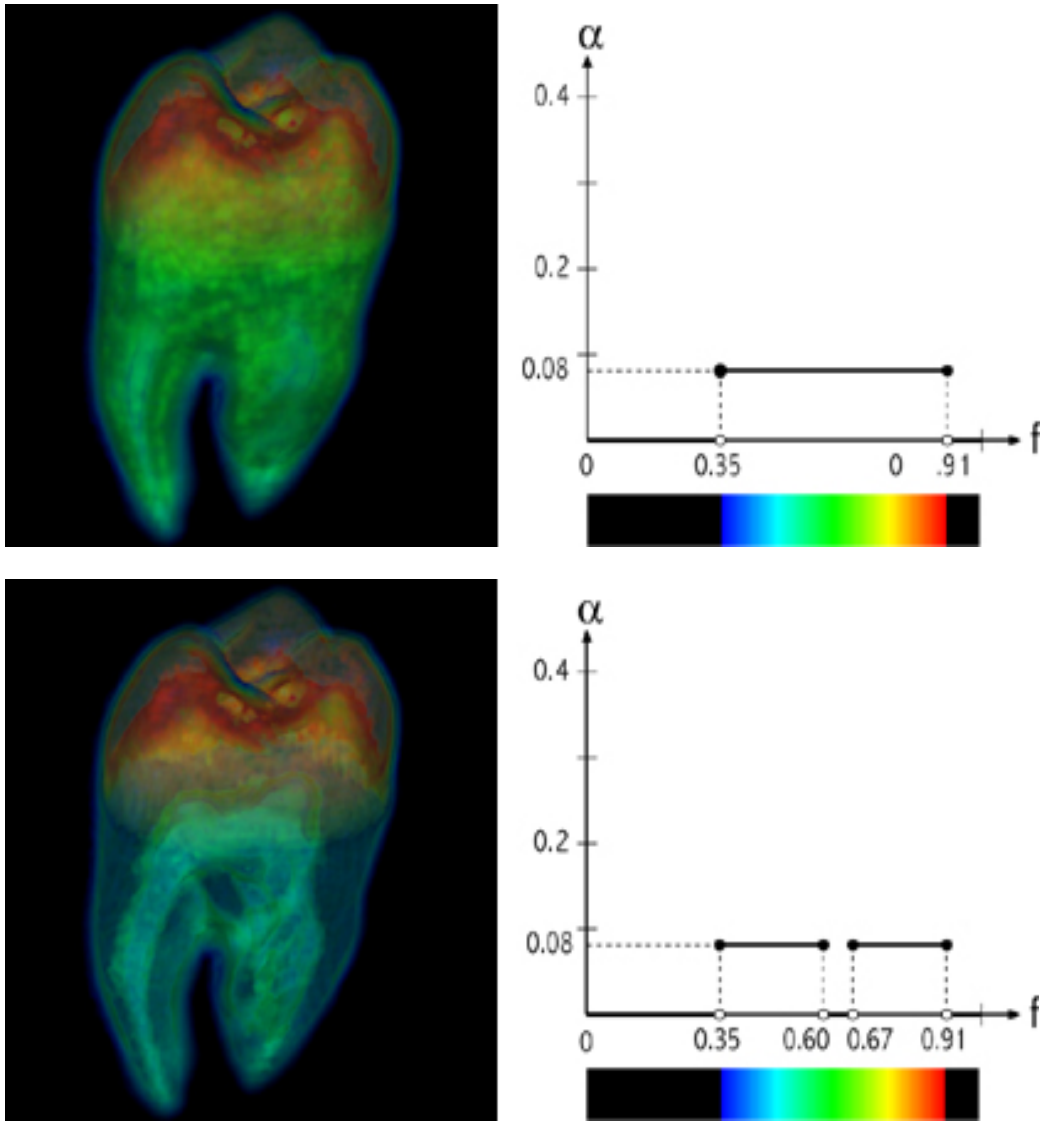
α不透明度

Transfer Function

- フィールド値をカラー情報と不透明度に変換
  - カラー情報　　任意
  - 不透明度
    - 各ボクセル密度（フィールド値）に応じて遮光効果が増加する
    - 密度勾配を考慮する
- 画像の品質に大きく影響
- TFの設計
  - 試行錯誤が必要

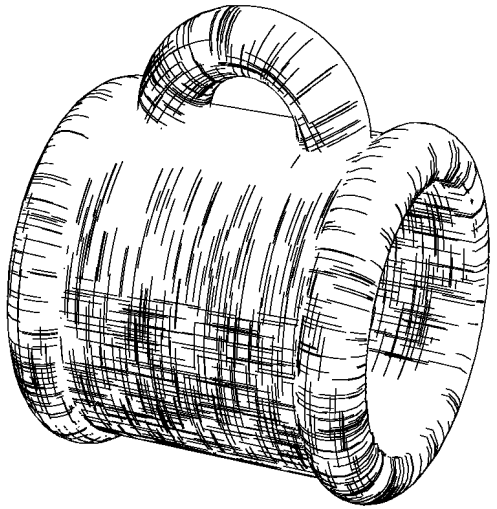


Transfer Functionの影響



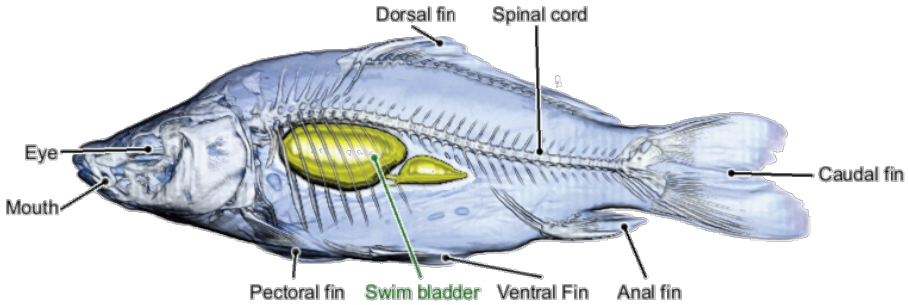
Non-Photo Realistic

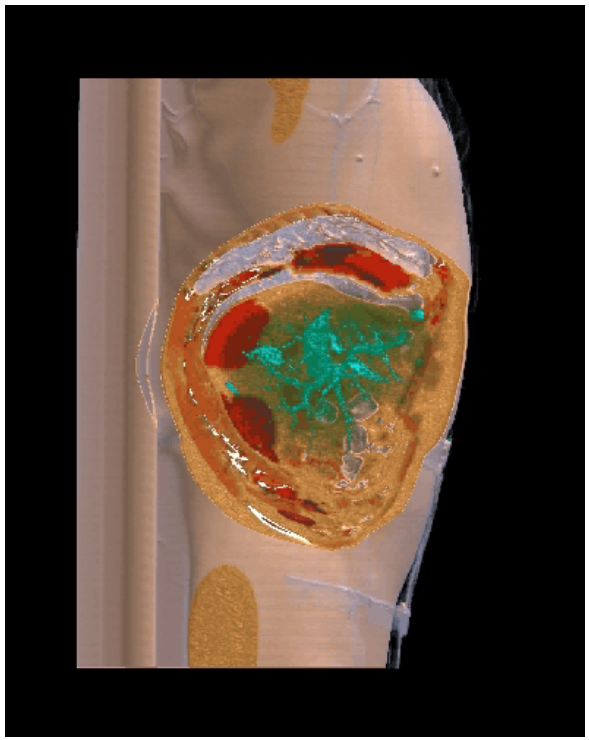
- 現象の特徴を自動的に抽出
  - 数学的処理
  - 特異点解析
- レンダリングテクニック
  - 素描
  - 絵画技法の模倣
- 特徴を際立たせる
  - S/N比の向上
  - 特徴・本質を描画
  - 特徴の抽出



http://www.mpi-sb.mpg.de/~ohtake/

Illustrative Rendering & Annotation

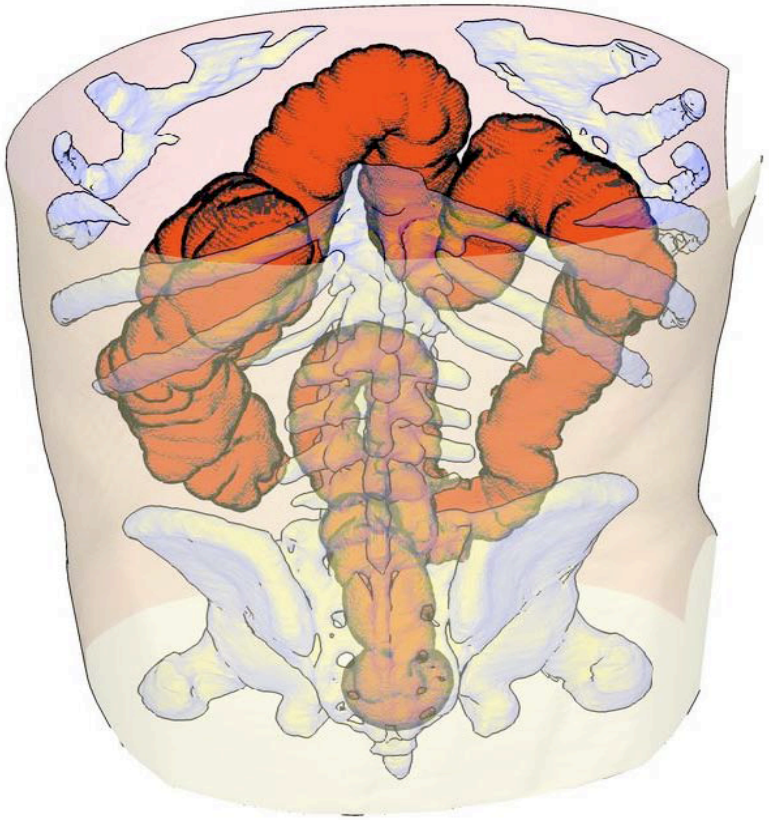




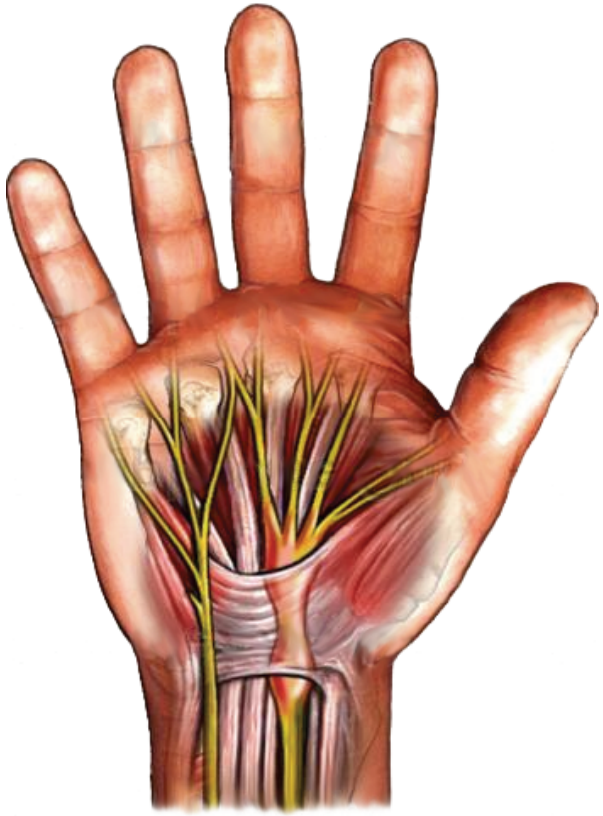
- Fully dynamic illustration
- Improve understanding
- Importance Driven

理解の補助

Simplification



Abstraction

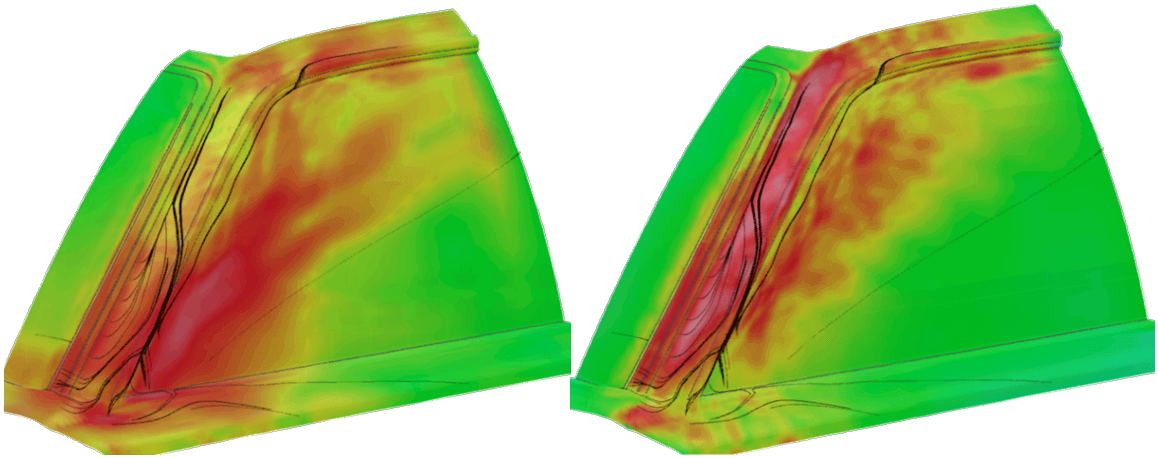


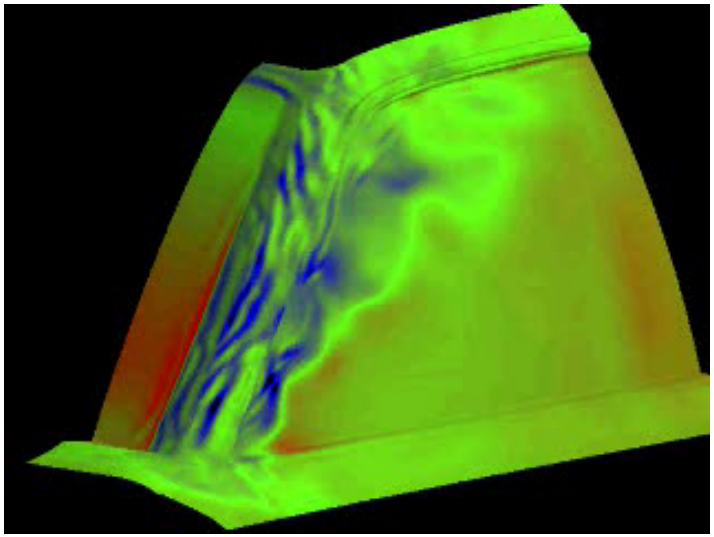
可視化技術

風音の分析と現象理解

車体表面の圧力の時間的な変動
=> 周波数分析を行い、各周波数域毎の特性をみる

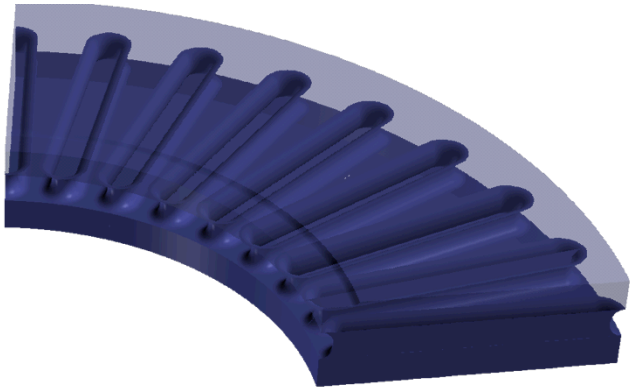
- 高周波の変動は局所的で、低周波変動は大きな渦の挙動に起因することがわかる





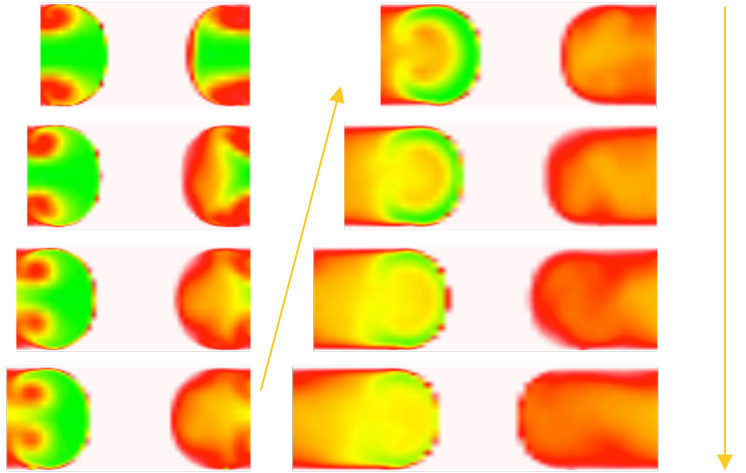
100Hz以下の変動　　100～500Hzの変動

ブレーキ冷却のアイデア



Cut model of rotor

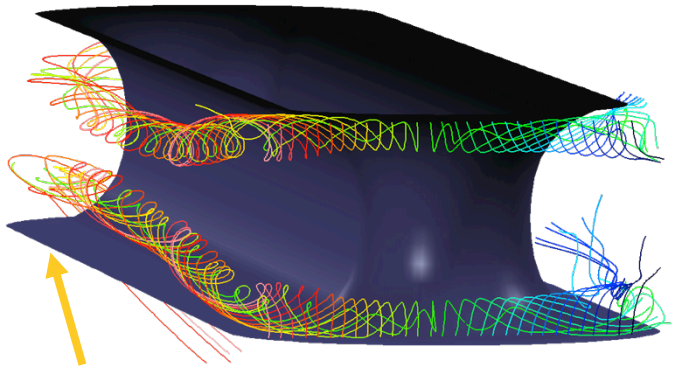
Inlet



Outlet

Temperature distribution
of cross-section
in ventilated hole

Outlet



Inlet

A pair of vortex tube
in cooling hole

Vortex enhances the heat transfer

可視化の数理的基礎

- 計算幾何
- 微分･積分方程式
  - LTE (Light Transport Equation)
- グラフ理論
- ベクトル解析
- 数値計算

- 認知心理学

可視化の難しさ

- 何をみたらよいか
- どのように見せたら良いか
- どういう手順で可視化するか
- システムをどうすればいいか

- 総合的な技術

可視化の多様性

- 100人いれば、100通りのやり方
- 様々な環境での利用
  - ローカルPCアプリ
    - PC環境（Mac, Linux, Win）
  - クライアント/サーバー方式
- タイミング
  - 計算終了後に可視化
  - 計算中に可視化
- データハンドリング
  - ファイル経由
  - オンメモリ
  - データ圧縮
- ユーザインタラクション
  - バッチ処理
  - インタラクティブ
- 対象データ
  - スカラ/ベクトル/テンソル
  - Time Varying
  - 多変量
- 計算機環境
  - PC、タブレット
  - クラスタ、スパコン

バッチ処理

- 計算機に処理を任せて，ジョブを投入
- 計算を終了するのを待つ
- 計算終了後に結果を確認
- 関連後処理を実施

インタラクティブ処理

- データに対し，ユーザが対話的に操作を行う
- 計算処理の反応がすぐに返る
- 思考しながら，データの中を探索できる

- 科学的発見に貢献

インタラクティブ性のためには？

- めちゃくちゃ速い計算機
  - CPU，ネットワーク，ディスク，GPU
- 現実的には，工夫が必要
  - データ量削減
  - 並列アルゴリズムによる効率化
  - Streaming処理

データ削減

- ROIの特定による観察領域の制限
  - そのためには、データを可視化する必要
  - あるいは、経験的に決定
- データ圧縮
  - 汎用の手法を適用 >> zlib, RLE, ...
  - 並列POD
  - JHPCN-DF
- 特徴抽出
  - 流体だったら、渦管、粒子

データ圧縮

- コンパクトな表現によりデータサイズを縮小
- Lossless圧縮
  - RLE（run length encoding）
  - zip
  - fpzip (Lindstrom)
- Lossy圧縮
  - Jpeg
  - Wavelet
  - POD（固有直交展開，固有値計算を利用）

- 大規模データには効率的な並列圧縮アルゴリズムが必要
- GPU実装も有望
- 圧縮，展開コスト，ファイルサイズ，メモリ消費量などがポイント

データ圧縮　JHPCN-DF

- Hagita, et al., “Efficient Data Compression by Efficient Use of HDF5 Format," SC14, 2014
- 論文をベースにライブラリとして実装
  - https://github.com/avr-aics-riken/JHPCN-DF
- データを用いて、ライブラリを評価中

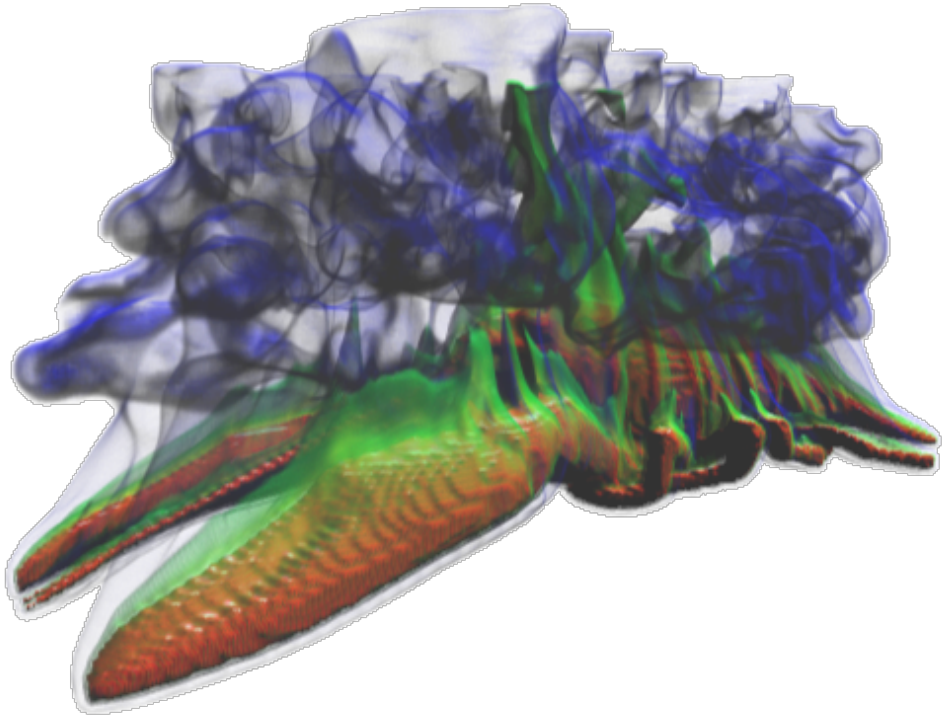
可視化システム技術

- 多くの種類のシミュレーション
  - 流体、構造、熱、粒子、分子・・・

- 多数のシミュレーションの可視化に使える、共通的な可視化システム
  - データ型に着目
    - スカラー、ベクトル、テンソル、パーティクルなど
  - 表現の共通化
    - 等高線、等値面、流線、粒子追跡、ボリュームレンダリング
  - データコンテナとしてのフォーマット
    - Vtk, UCD, netCDF, HDF5 ⇔ 独自フォーマット
  - インタラクション
    - ユーザの意図に応じて可視化の手順を指示
  - カスタマイズ
    - 特定目的のための機能化

既存の可視化システム

- 商用
  - AVS
  - FieldView
  - EnSight
  - VisLink
  - RVSLIB

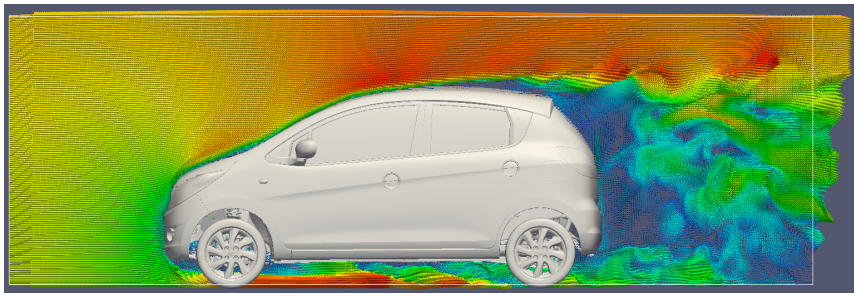


- 非商用
  - Visit https://wci.llnl.gov/codes/visit/
  - ParaView http://www.paraview.org/
  - V-Isio http://avr-aics-riken.github.io/V-Isio/
  - HIVE http://avr-aics-riken.github.io/HIVE/

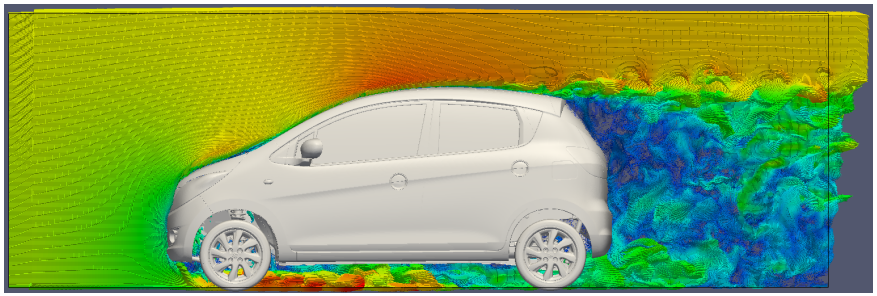
大規模並列計算

大規模計算の一例

- 必要性
  - 量的なアプローチで、これまで出来なかったことが可能になる
    - 時間のかかっていた計算が短時間でできるようになる
    - シミュレーション結果の信頼性が高くなる
    - 複雑な物理モデリングを取り入れることが可能になる
- ビッグデータ
  - 定義：Volume, Variety, Velocity,…
  - Scientific Vis. >> 規模が重要（多様性も少し）、データは構造化

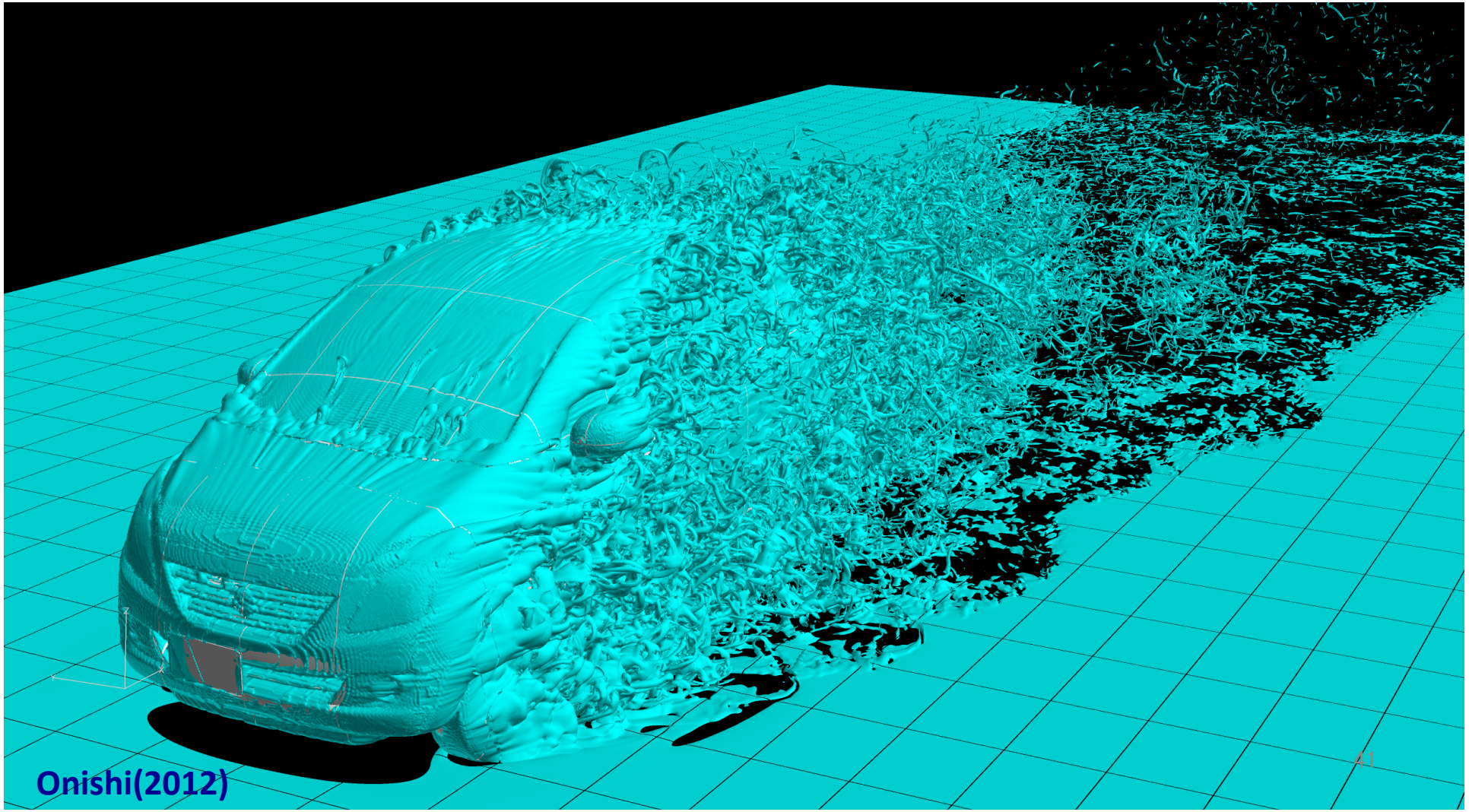


4.5億格子（9GB/5変数）

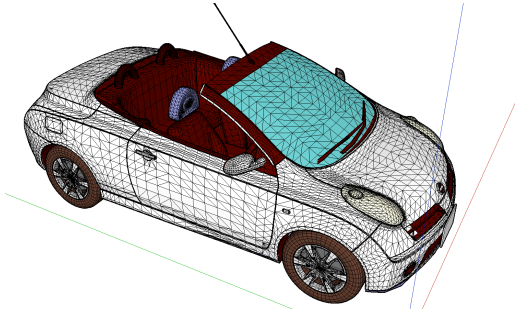


36億格子（72GB/5変数）

Vortex Structure on 30Billion Grids



例えば，車の計算

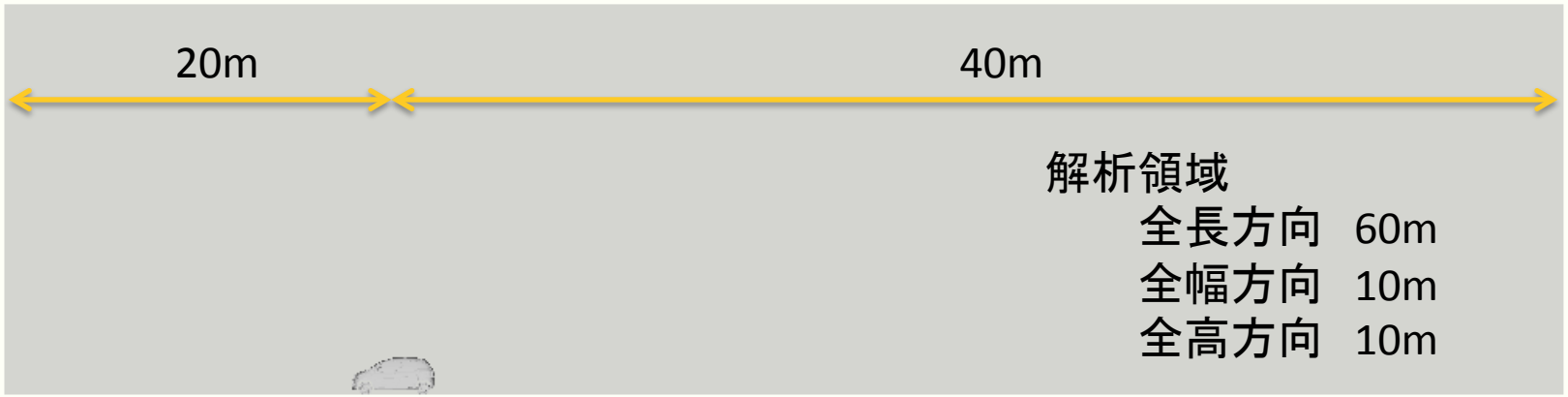


全長　3.4m
全幅　1.6m
全高　1.5m

前面投影面積 2.4 m^2

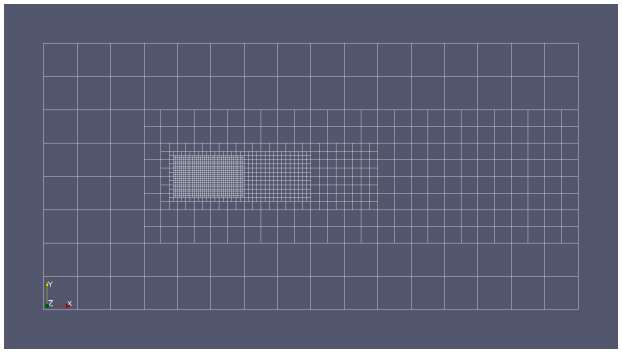
ブロッケージ比=車の断面積/測定区間の断面積 = 0.2%

走行速度　　108 km/h = 30 m/s
境界層内の細かい渦のスケール　100 μm ~ 格子解像度 0.1mm
600000 x 100000 x 100000 = 6 x 10^{15} !!
時間積分幅 Δt = 3.3 x 10^{-7} 秒

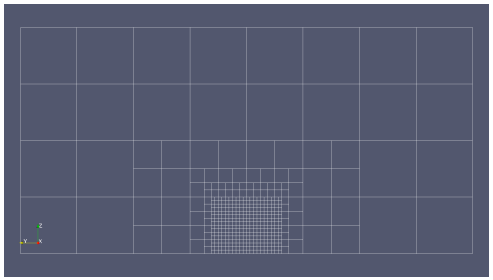


階層型格子による空力解析例

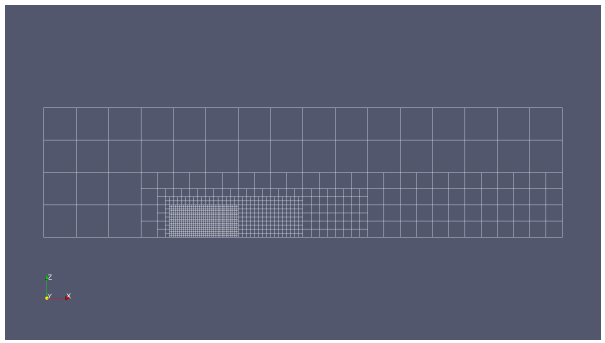
- 階層型直交格子
- 最大レベル数：6
- 総格子数：約4.5億
- 最小格子間隔：4[mm]



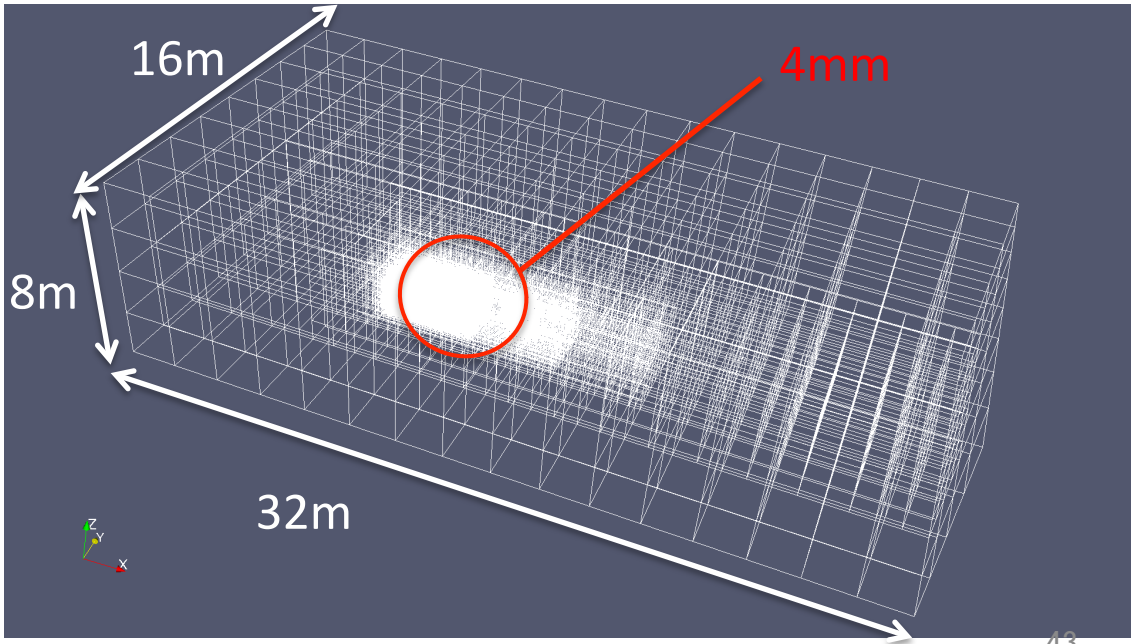
■上面



■正面



■側面



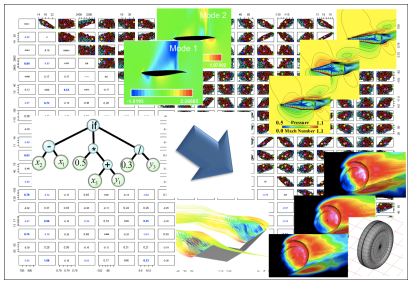
■3D

音の計算の場合

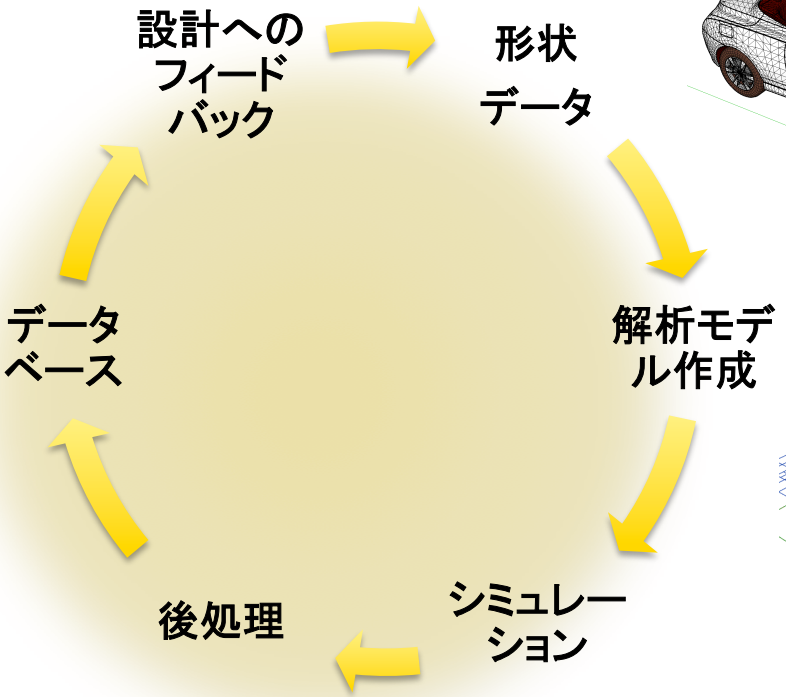
- 300億セルで精密な計算を行う
  - 10~2kHzあたりの評価のためのサンプリング
    - 低周波側の位相平均を20回 >> 定常2秒間の計算
    - 高周波側 >> サンプリング定理 $\Delta\tau=1/(2f)=2.5\times10^{-4}$秒
    - ２秒間を$\Delta\tau$でサンプリング >> 8000 time slice

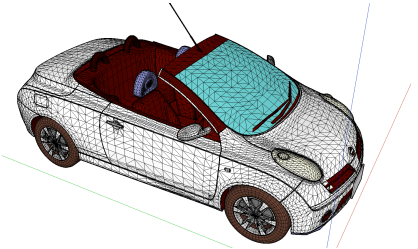
- 必要諸量
  - 単精度（4bytes）で出力 >> 30G x 4B = 120GB (scalar)
  - 変数は，速度(3)，圧力(1)，温度(1) >> 120GB x 5 = 600GB
  - 600GB x 8000 slices = 4.8PB
  - 10000プロセスでの計算 (file-per-process)
    - ファイル数=10000 x 4vars x 8000 slices = 3.2×10^8 files

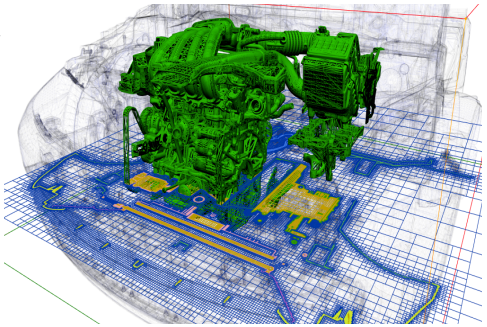
シミュレーションのサイクル



JAXA Oyama







大規模並列計算の後処理における問題点

- データコピー，移動が高コスト
  - 処理時間，MMU/HDD容量
- データは動かせない
  - 適切なツールなしにはデータアクセスさえ不可
  - 動かせたとしても，処理手続きが煩雑，面倒
- 要因
  - データの大規模性
    - 空間規模，時系列データ，多変数
  - データの分散性
    - 分散並列，GRID
  - システムの複雑性
    - ヘテロ環境，ファイルシステム，ネットワーク

大規模データ可視化のチャレンジ

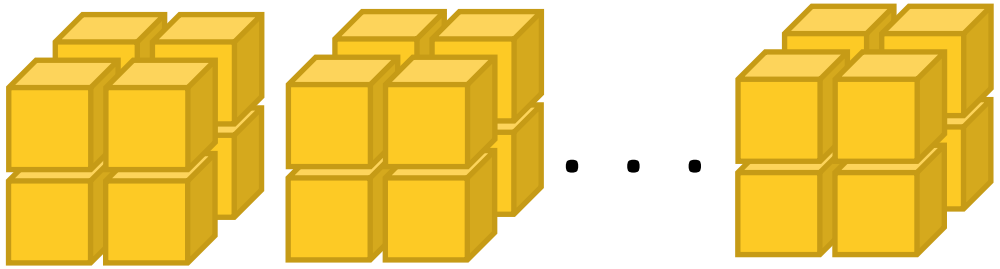
- チャレンジ
  - スケーラブルなインタラクティブ可視化
  - 大規模データのレンダリングとデータ分析の融合

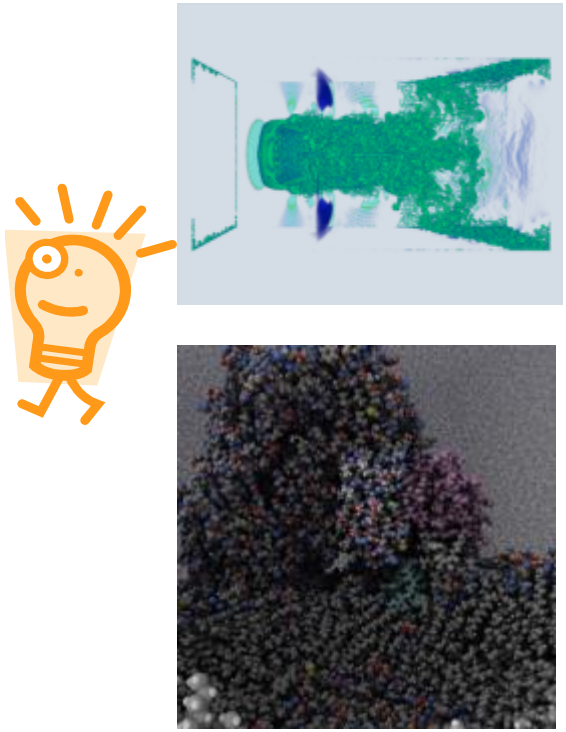
- 要素研究
  - 並列データ圧縮
  - ビッグデータ向け分析アルゴリズム
  - ワークフロー
  - データマネジメント、DB
  - ストリーミング
  - レンダリング
  - 多変量可視化
  - 非定常可視化

並列レンダリング

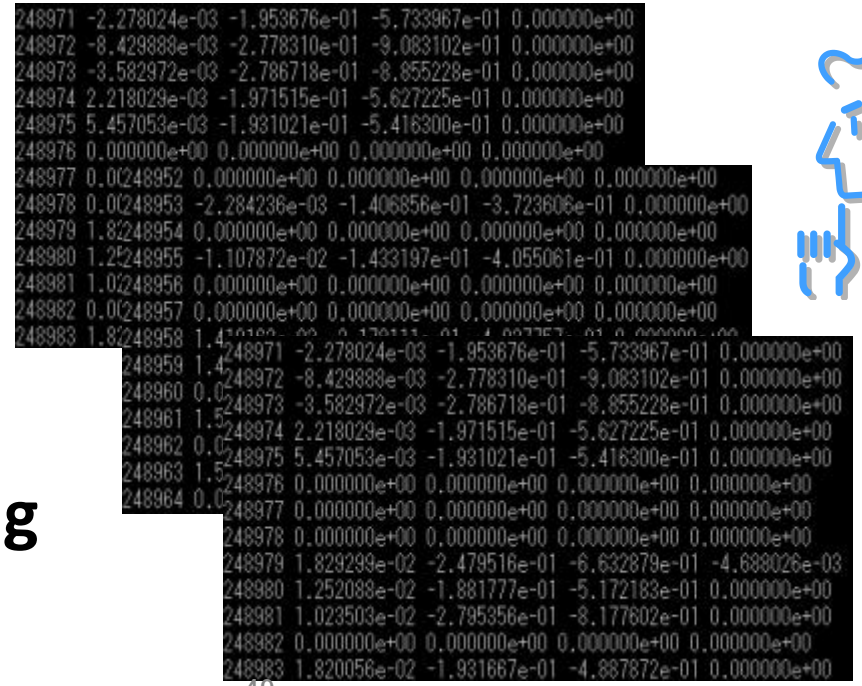
Scientific Visualization

Numerical Simulation Results



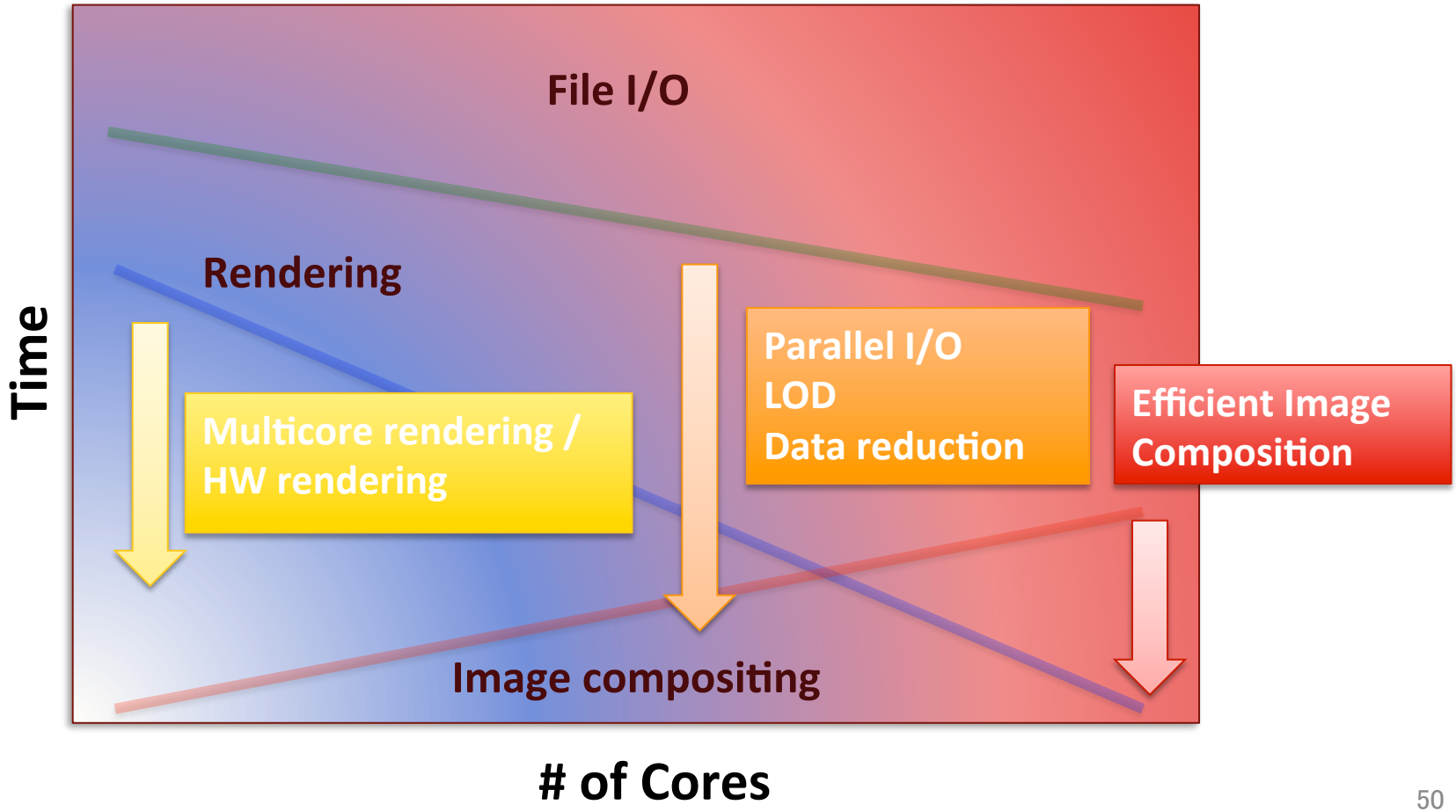


Rendering

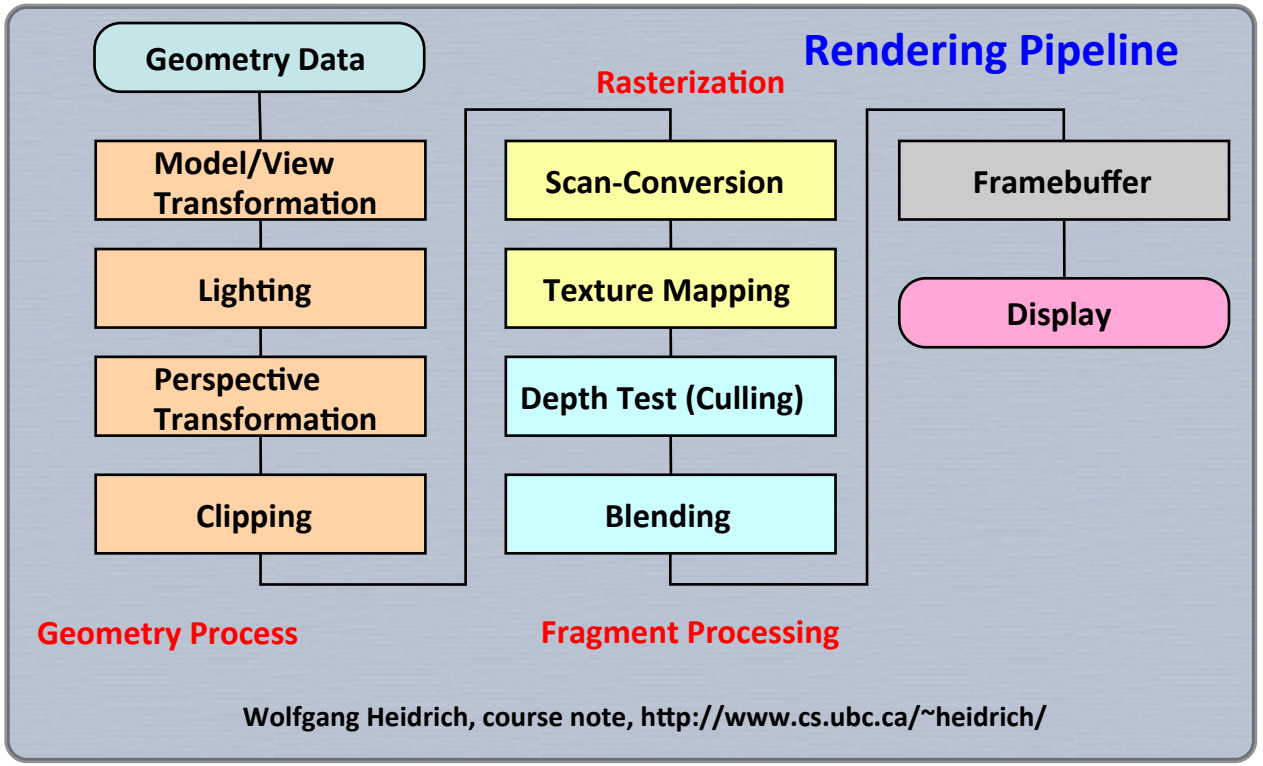


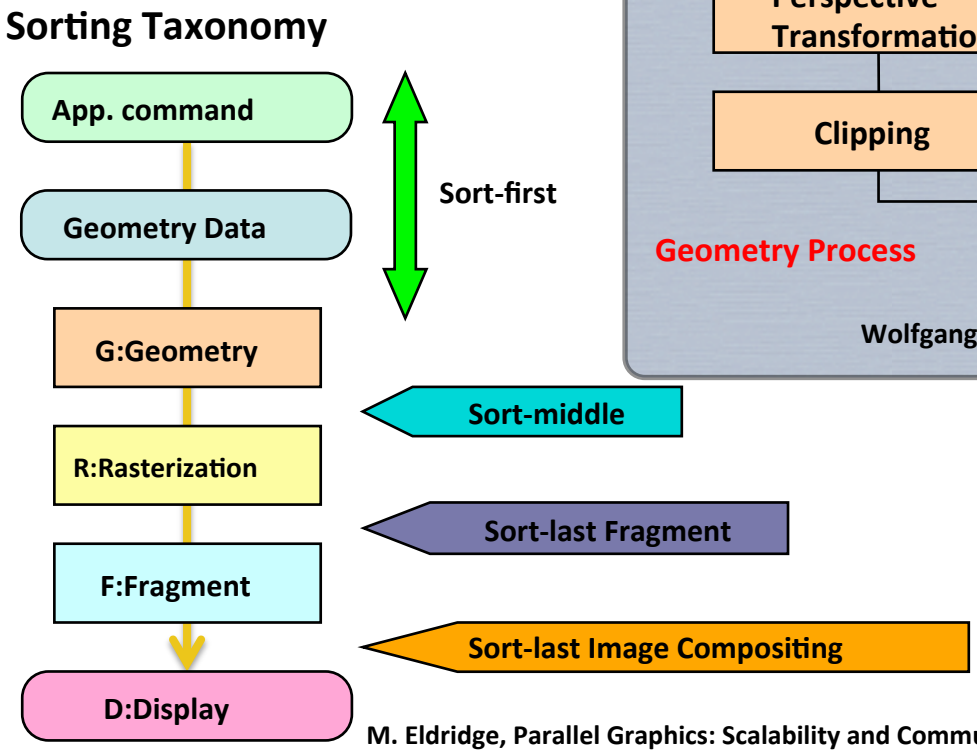
K computer

スケーラビリティとインタラクティブ性の戦略



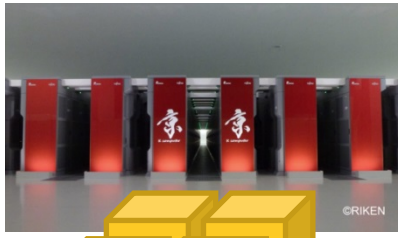
Graphics pipeline in parallel rendering





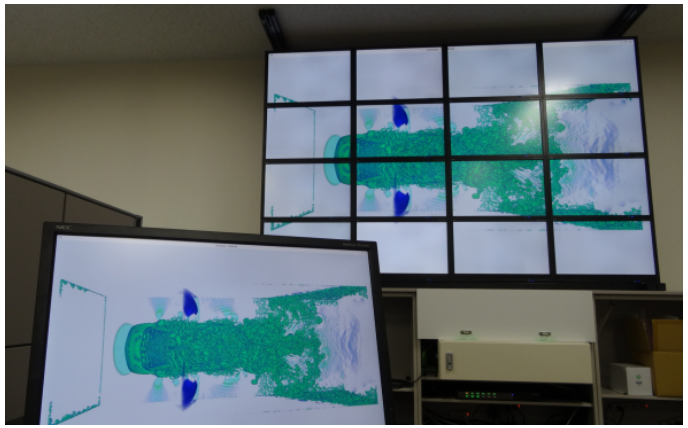
M. Eldridge, Parallel Graphics: Scalability and Communication, Course Note 37, SIGGRAPH 2001.

Sort-Last Parallel Rendering



Parallel Rendering

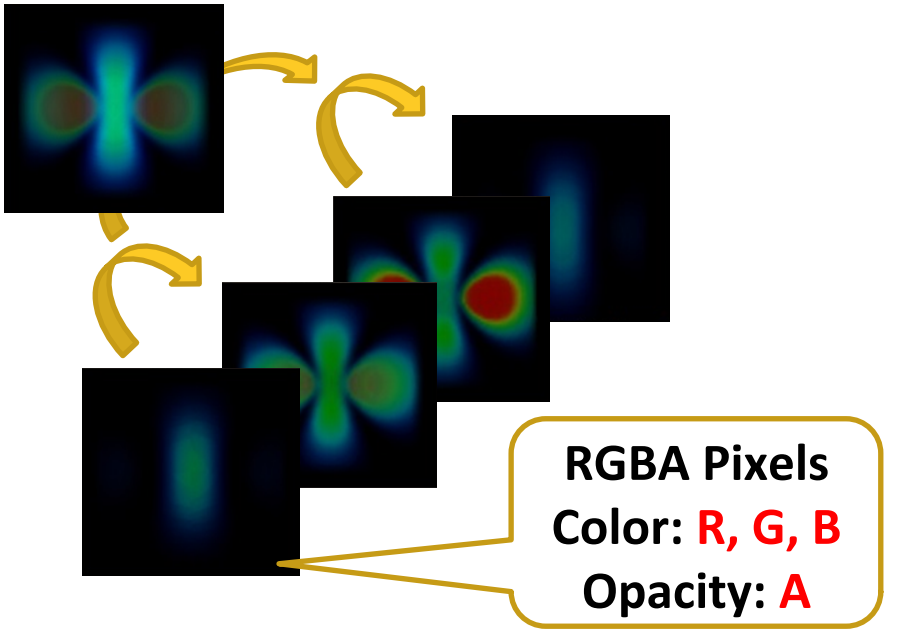
Parallel Image Composition

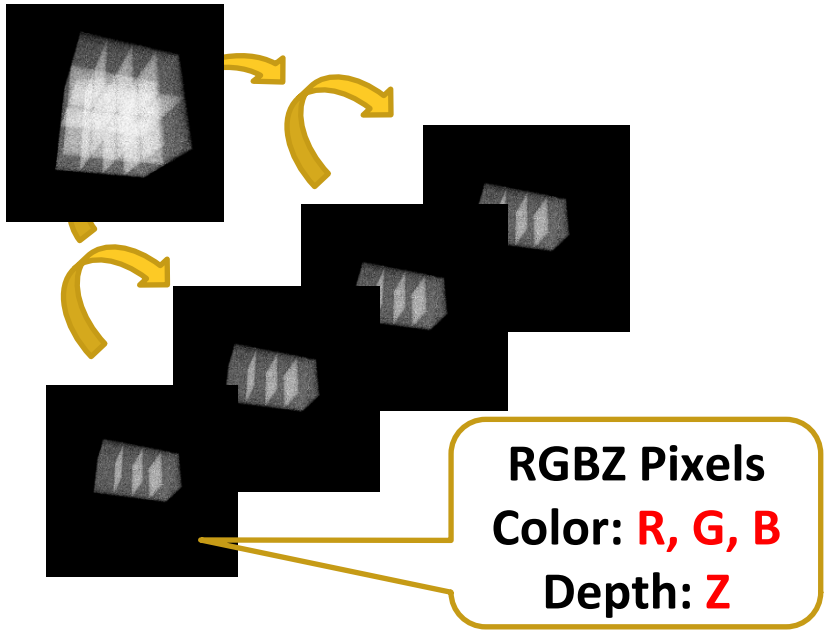


K computer

Image Composition

- Per-pixel Image Processing





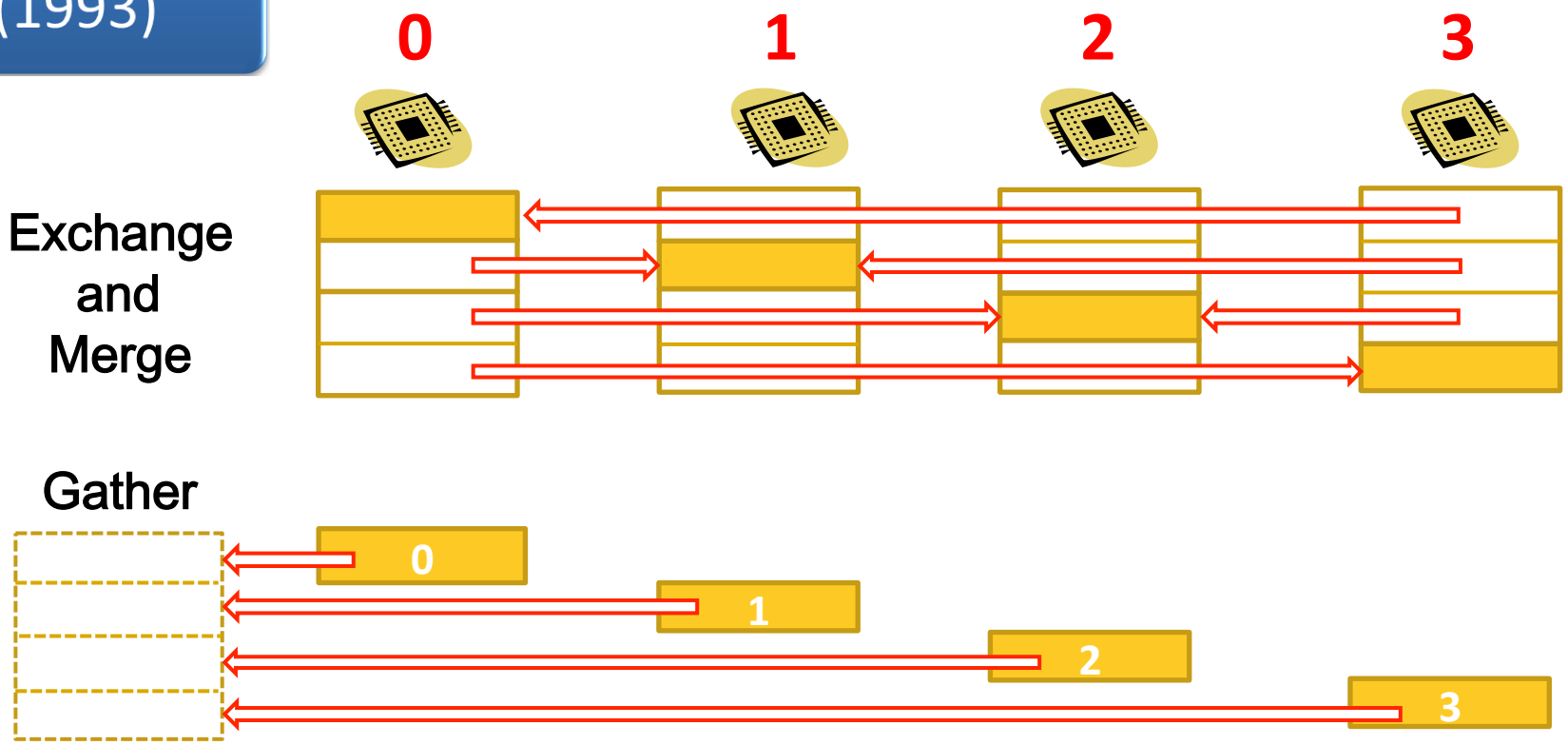
Parallel Image Composition

Communication + **Computation**

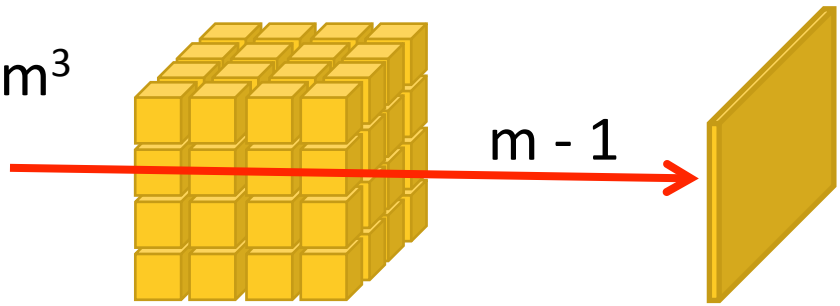
(Send/Receive Pixels) **(Alpha Blending/Z Comparison)**

Direct-Send (1993)

Communication

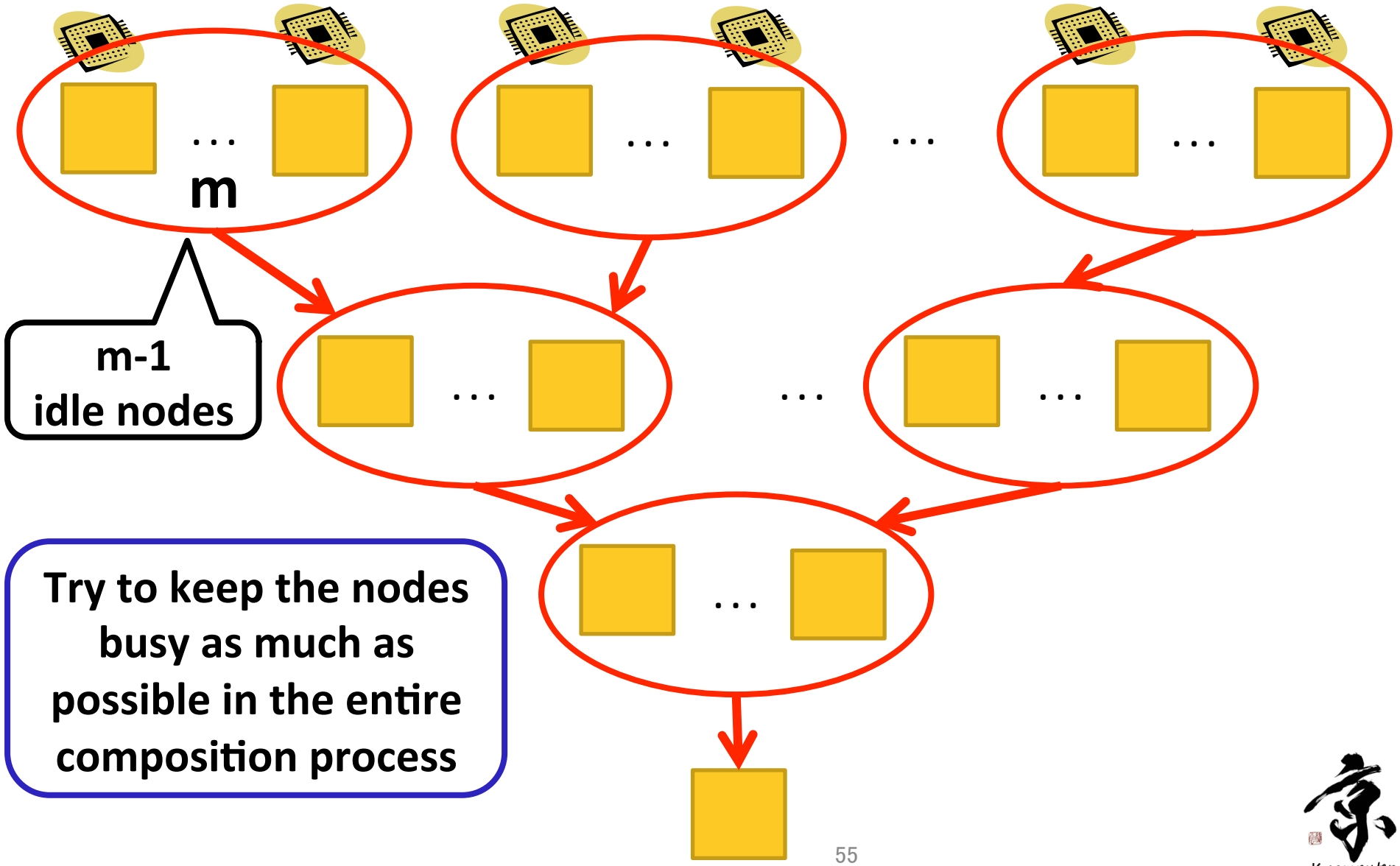


Ex.: $n = m^3$



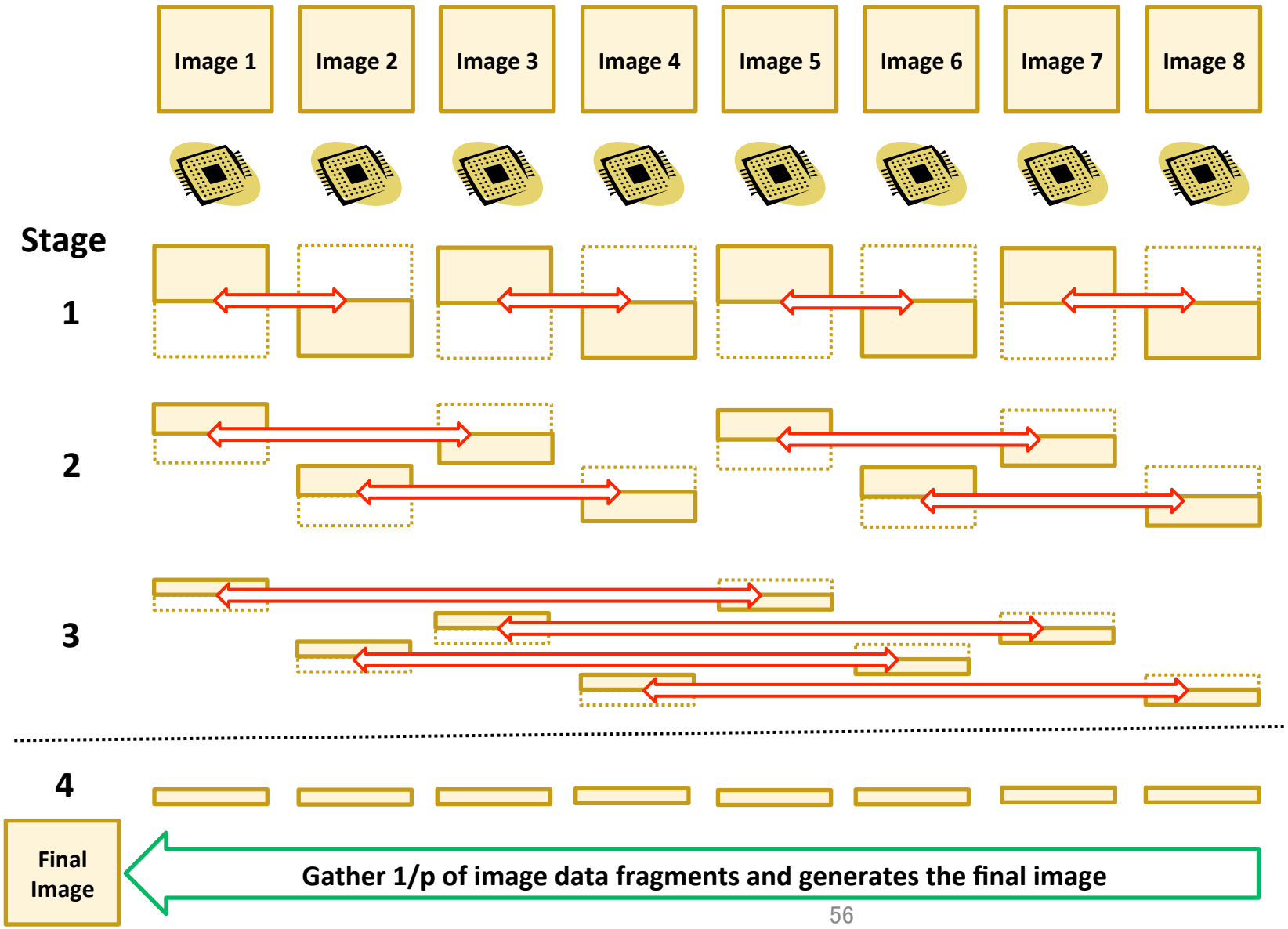
K computer

Communication



K computer

Binary-Swap Image Compositing

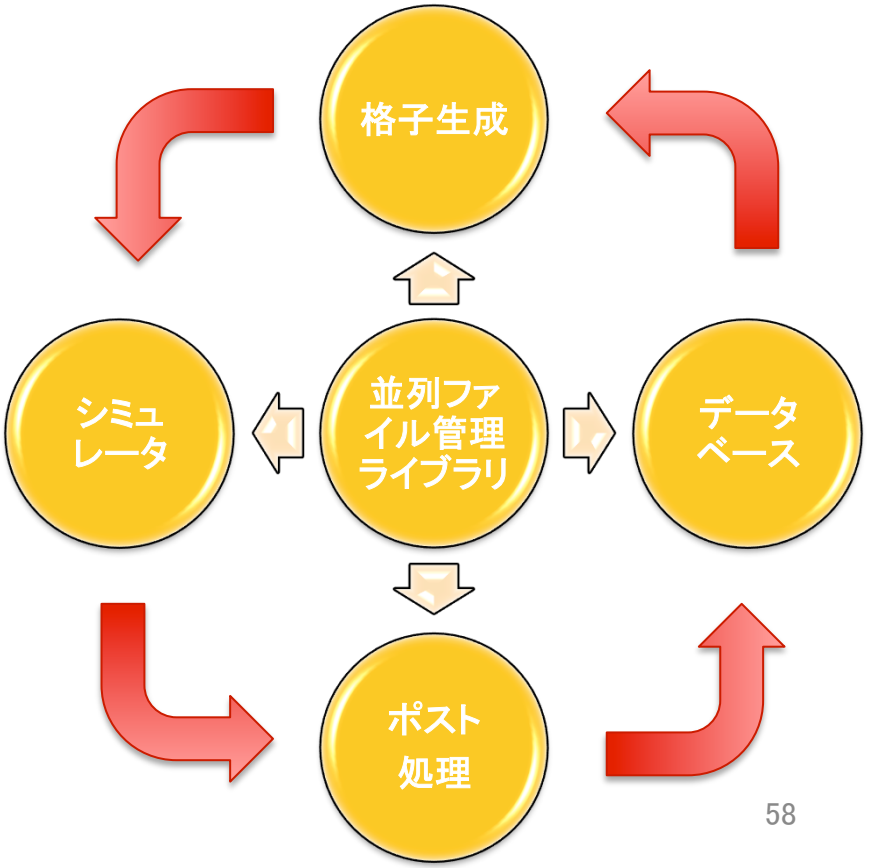


K computer

並列分散ファイル管理

ファイル管理は要

- 解析サイクルを回す上でデータ管理は重要な技術
  - 特に、大規模データを生成するシミュレータとポスト処理の間で多数のファイルが存在
- ファイル管理の役目
  - 多数のファイルの存在（実体）を隠蔽
  - メタファイルによる管理
- ファイルシステムとの関連
  - ローカルファイルシステム
  - グローバルファイルシステム

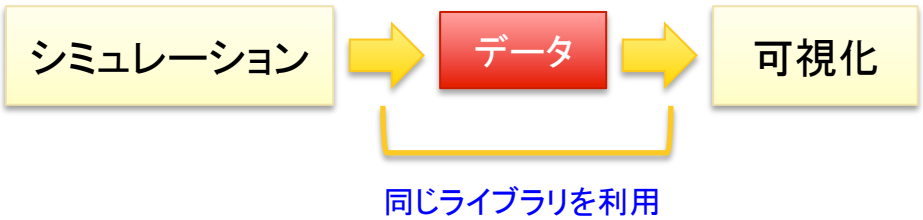


並列データ管理　xDMlib

- ファイル数：変数　×　並列数　×　タイムステップ
- ファイル管理のためのメタデータ設計
  - 多数のファイルを管理することが主眼
- ファイルI/Oのポリシは、File per node
  - 性能はハードウェアとミドルウェア任せ
  - 「京」のファイルシステムは比較的高性能

X is ;
- **C 構造データ**
- **U 非構造データ**
- **P 粒子データ**

- 既存ライブラリ
  - netCDF, CGNS, Vtk, ADF, Siloなど
    - データコンテナとして利用可能
    - 利用のための学習コスト
  - HDF5
    - 高並列時に高性能、使うには沢山のパラメータ調整が必要
    - 圧縮など、ユーザ側で工夫したい場合には空間結合しない使い方が扱いやすい場合もある

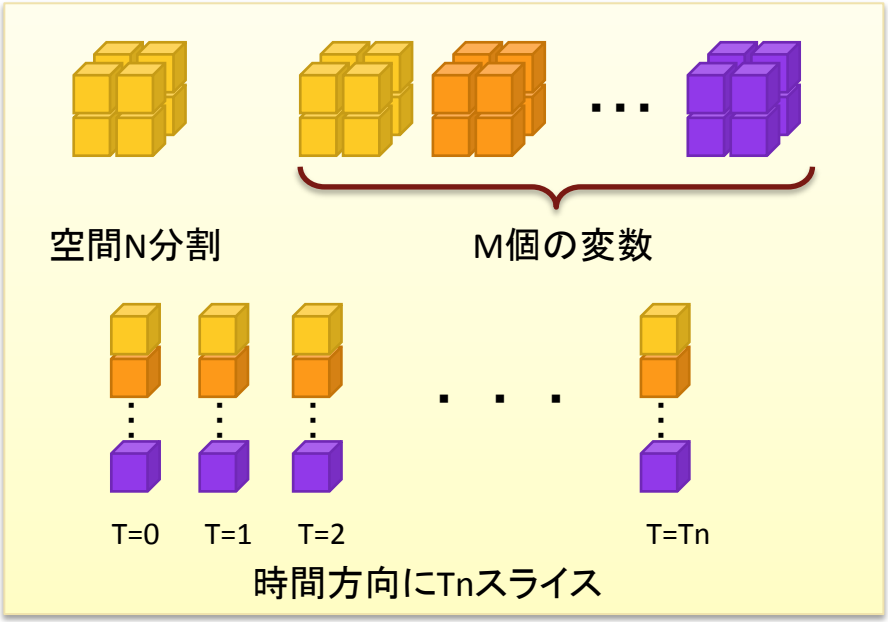


メタデータ

- プロセス毎にファイルを生成
- 各プロセス固有の情報をメタファイルで表現し、ファイルの実体とリンクさせる
- メタファイルを読むことで、データの概要を理解できる
  - データの実体である多数のファイルはディレクトリに保存したまま触らない
  - 軽量なメタデータのみを参照し、データを操作
- メタファイル
  - インデクスファイル　index.dfi （データコンテナ毎に1つ）
  - プロセスファイル　　proc.dfi

ファイル管理のパターン

- パラメータ
  - N　　並列数（領域分割数）
  - M　　変数
  - T　　タイムスライス



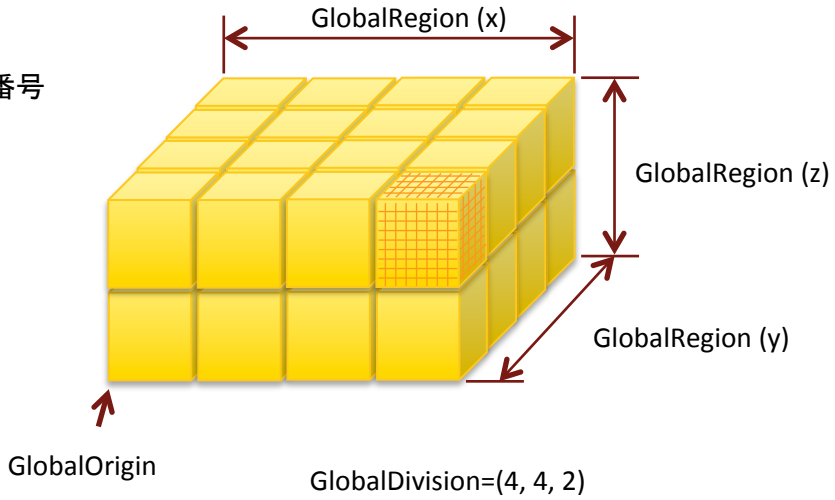
- 生成ファイル数のパターン
  1. N x M x T　　各パラメータ毎にファイルを生成
  2. N x T　　変数を一つにまとめる
  3. N　　1プロセス毎に1ファイルとする
  4. T　　時刻毎に1ファイルとする
  5. M x T　　空間を一つに結合する
  - パターンによって、1ファイルの大きさとファイル数が異なる　=> システム依存

CDMlib :: Proc.dfi

```
// ドメイン情報
Domain {
 GlobalOrigin   = (-3.00, -3.00, -3.00) // 計算空間の起点座標
 GlobalRegion   = ( 6.00,  6.00,  6.00) // 計算空間の各軸方向の長さ
 GlobalVoxel    = (64, 64, 64)          // 計算領域全体のボクセル数
 GlobalDivision = (1, 1, 1)             // 計算領域の部分領域の分割数
}

// 並列情報
MPI {
  NumberOfRank  = 128  // プロセス数
  NumberOfGroup = 1    // グループ数
}

// 各プロセスの情報
Process {
  Rank[@] {
    ID         = 0                  // ファイル出力時のランク番号
    Hostname   = "Iridium.local"    // 計算ノードのホスト名
    VoxelSize  = (64, 64, 64)       // 各領域のボクセルサイズ
    HeadIndex  = (1, 1, 1)          // 各領域の開始インデクス
    TailIndex  = (64, 64, 64)       // 各領域の終了インデクス
    CellID     = 1                  // 格子データ確認用フラグ
    BCflagID   = -1                 // 境界条件確認用フラグ
  }
  ...
}
```



CDMlib :: index.dfi

```
// ファイル情報
FileInfo {
  DFIType       = "Cartesian"
  DirectoryPath = "./hoge"   // ファイルの存在するディレクトリ
                             （DFIファイルからの相対パス）
  TimeSliceDirectory = "off"
  Prefix        = "vel"      // ベースファイル名      ※1
  FileFormat    = "sph"      // ファイルタイプ、拡張子 ※1
  FieldFilenameFormat= "step_rank"                    ※1
  GuideCell     = 0
  DataType      = "Float32"  // データタイプ          ※2
  Endian        = "little"   // データのエンディアン   ※3
  NumVariables  = 3          // 成分数(スカラーは1)   ※4
  Variable[@] { name = "U" } // 各成分の変数名
  Variable[@] { name = "V" }
  Variable[@] { name = "W" }
}


// プロセス情報のファイルパス
FilePath {
  Process      = "proc.dfi"
}
```

```
// 単位系 必要に応じて追加
Unit {
  Length   = "M"   //(NonDimensional, m, cm, mm)
  L0       = 1.0   // 規格化に用いた長さスケール
  Velocity = "m/s" //(NonDimensional, m/s)
  V0       = 3.4   // 代表速度 (m/s)
  Pressure = "Pa"  // (NonDimensional, Pa)
  P0       = 0.0   // 基準圧力(Pa)
  DiffPrs  = 510.0 // 圧力差(Pa)
  Temperatur = "C" // (NonDimensional, C, K)
  BaseTemp = 10.0  // 指定単位
  DiffTemp = 35.0  // 指定単位
}
```

```
※1 ファイル名
[Prefix]_[ステップ番号:10桁]_id[RankID:6桁].[Extension]

※2 Int8, UInt8, Int16, UInt16, Int32, Uint32, Int64,
UInt64, Float32, Float64

※3 little, big, 省略時:実行プラットフォームと同じ
```

index.dfi (contd.)

```
// 時系列情報
TimeSlice {
  Slice[@] { // ファイル出力回数分
    Step = 0
    Time = 0.0
    MinMax[@] {         // NumVariables個
      Min  = -1.56e-2
      Max  = 8.2e-01
    }
    ... 任意のアノテーションが追加可能
 }
  ...

  Slice[@] {
    Step = 1000
    Time = 10.0
    MinMax[@] {         // NumVariables個
      Min  = -8.5e-1
      Max  = 9.1e+01
    }
    ... 任意のアノテーションが追加可能
 }
}
```

HIVE System

背景

- 高性能計算＝大規模並列計算
  - 多量のデータを生成する。しかも、分散ファイル

  ⇒高性能な並列可視化
- もはやデータは動かせない
  - データのあるところで絵を作る

  ⇒リモート可視化
- 試行錯誤が発生する
  - 可視化のパラメータを決めるのは時間がかかる

  ⇒インタラクティブ可視化
- 可視化は多くのシナリオ（やり方）がある
  - 作業者や環境に応じて、カスタマイズしたい

  ⇒カスタマイズ性
- 多くの計算機環境で、どこでも使いたい

  ⇒マルチプラットフォーム
- 長期にわたって利用したい
  - アーキテクチャへの依存性を小さく

  ⇒最新のソフトウェア技術

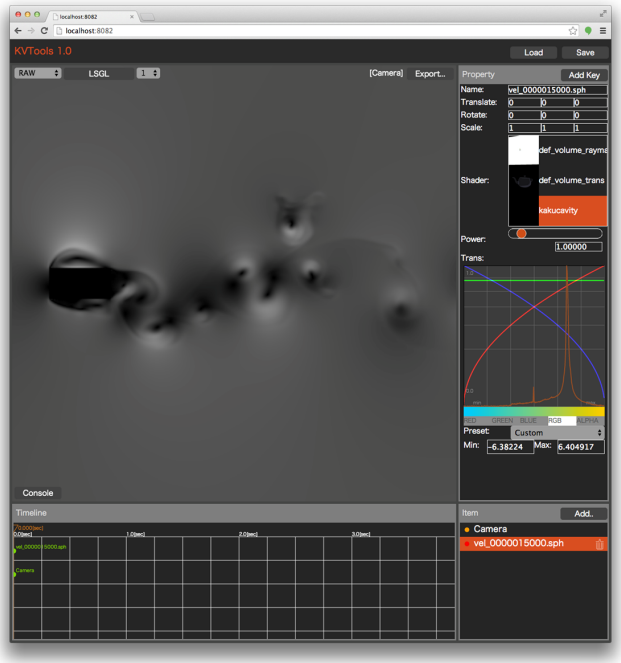
HIVEで考えるシナリオ

- リモート
- インタラクティブ
- バッチ
- スケーラブル
- マルチプラットホーム
- ユービキタス
- トランス・ジェネレーション

HIVE

- Heterogeneously Integrated Visual analytic Environment

- 大規模なデータを対象に、並列で効率よく可視化できる環境
  - {並列, リモート,インタラクティブ}可視化
  - データ分析との連携
  - Webブラウザ環境で可視化対象データのレイアウトや質感などをインタラクティブに調整できる
  - マルチプラットフォームで動作するスタンドアローンのレンダリングプログラム、HIVE render (hrender) に対応



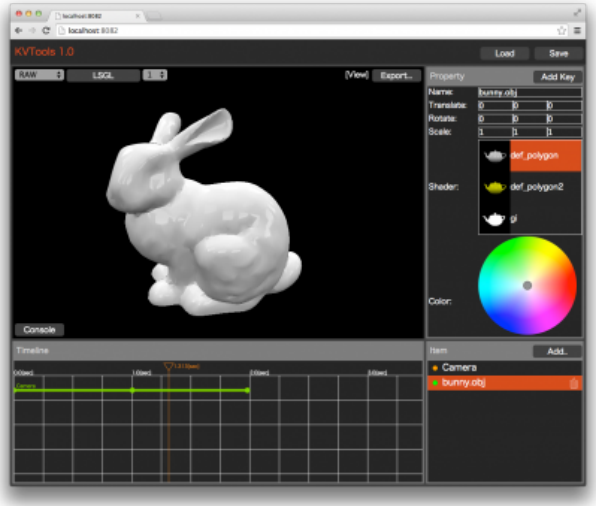
並列可視化の利用技術

- データロード
  - 並列分散ファイル管理ライブラリの利用（シミュレータと連携）
  - 並列にデータをロード
  - データ圧縮の併用

- レンダリング
  - 高品質なレイトレーサー、ボリュームレイキャスター
  - OpenGLES, GLSLの利用　⇒モバイルからスパコンまで
  - 大規模なピクセルイメージを生成可能（8k, 16k, 32k…）

- イメージ重畳
  - 2-3-4 Compositionを開発
  - 大規模並列で高速なデータ収集アルゴリズム

実装方式

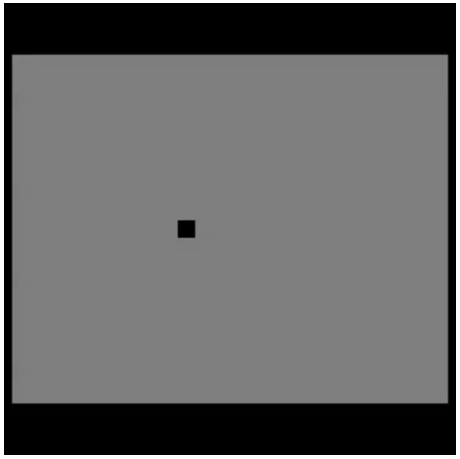


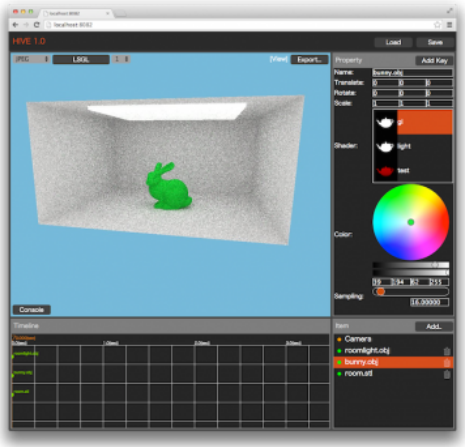
- マルチプラットフォーム
  - 軽量Webサーバーであるnode.js上で動作
    - http通信によるネットワーク透過性
  - OpenGLES, GLSLのレンダリングAPI
    - シェーダによるカスタマイズ性
- SURFACEレンダラ
  - 多くのプラットフォームとアーキテクチャで動作し、マルチスレッド、マルチCPUに対応したレンダリングライブラリ
- 最新のソフトウェア技術
  - メンテナンス性

レンダラー

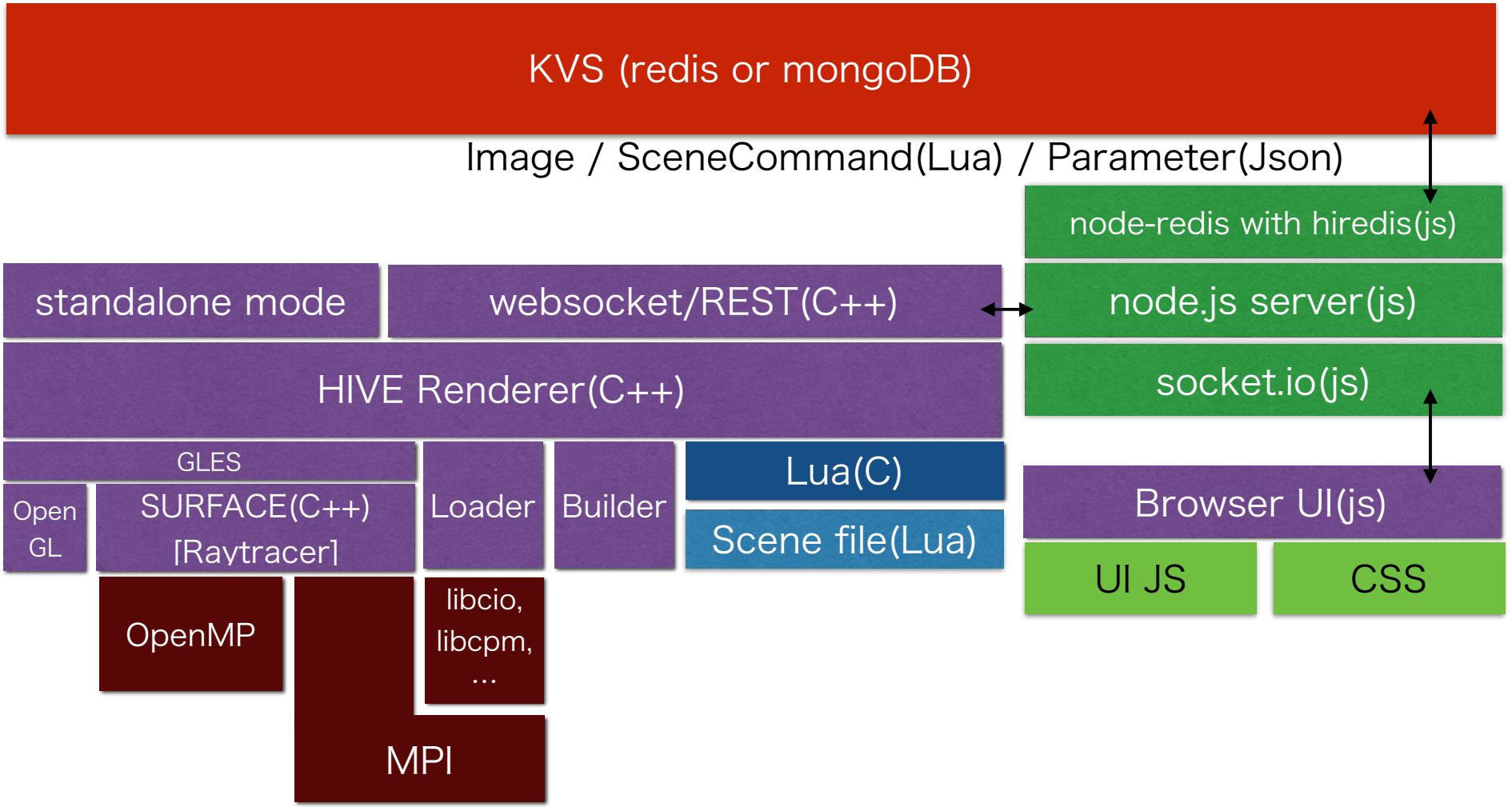
- 組み込み済みレンダリング
  - ポリゴンデータのグローバルイルミネーションレンダリング
  - ボリュームデータのレイマーチン法によるボリュームレンダリング
  - ボリュームデータの等値面レンダリング
  - ポイントデータの球形表現でのレンダリング
  - ラインデータの円柱形表現でのレンダリング

- GLSLベースのシェーダー言語を利用して質感を設定可能



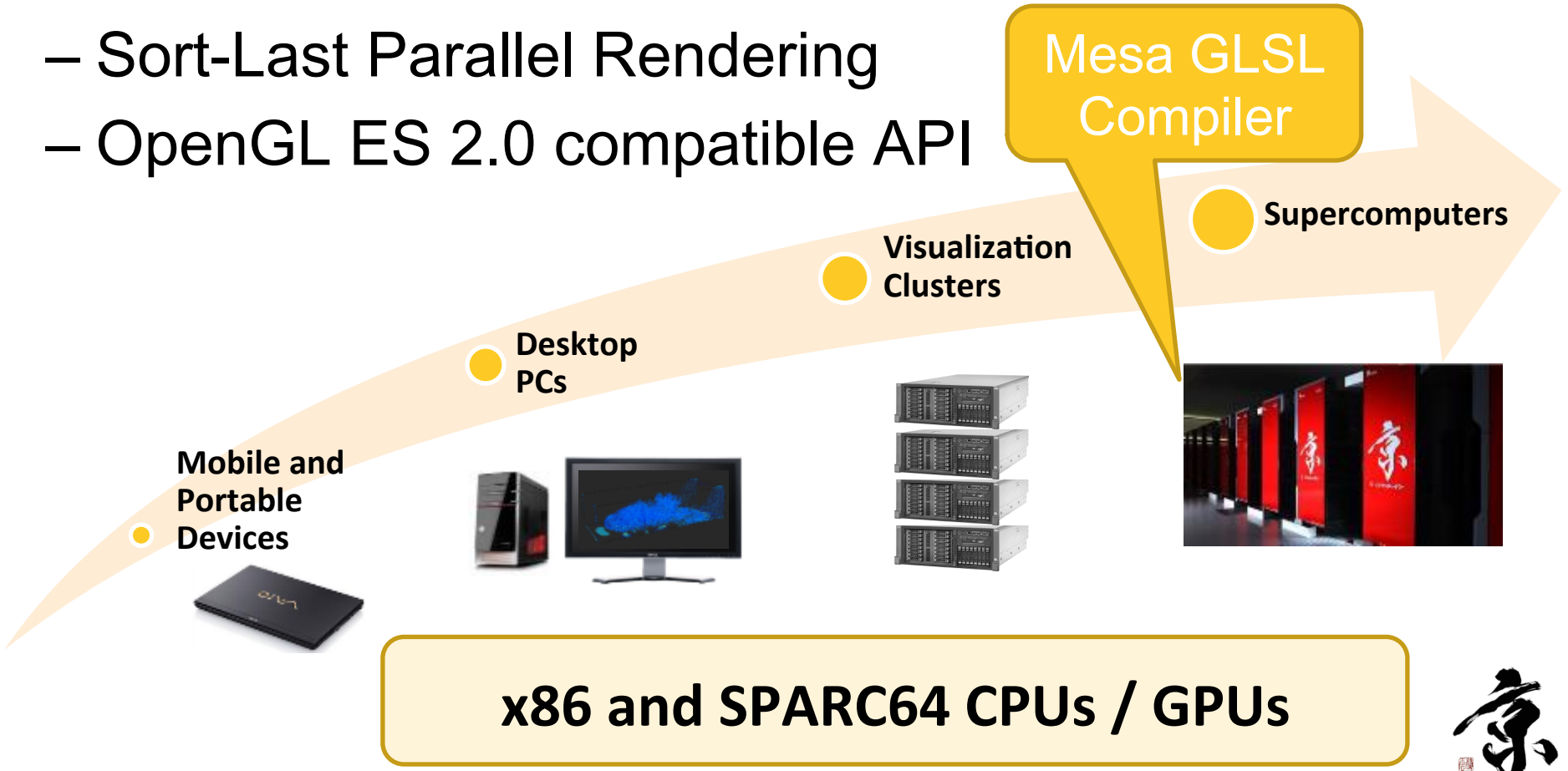


ソフトウェアスタック

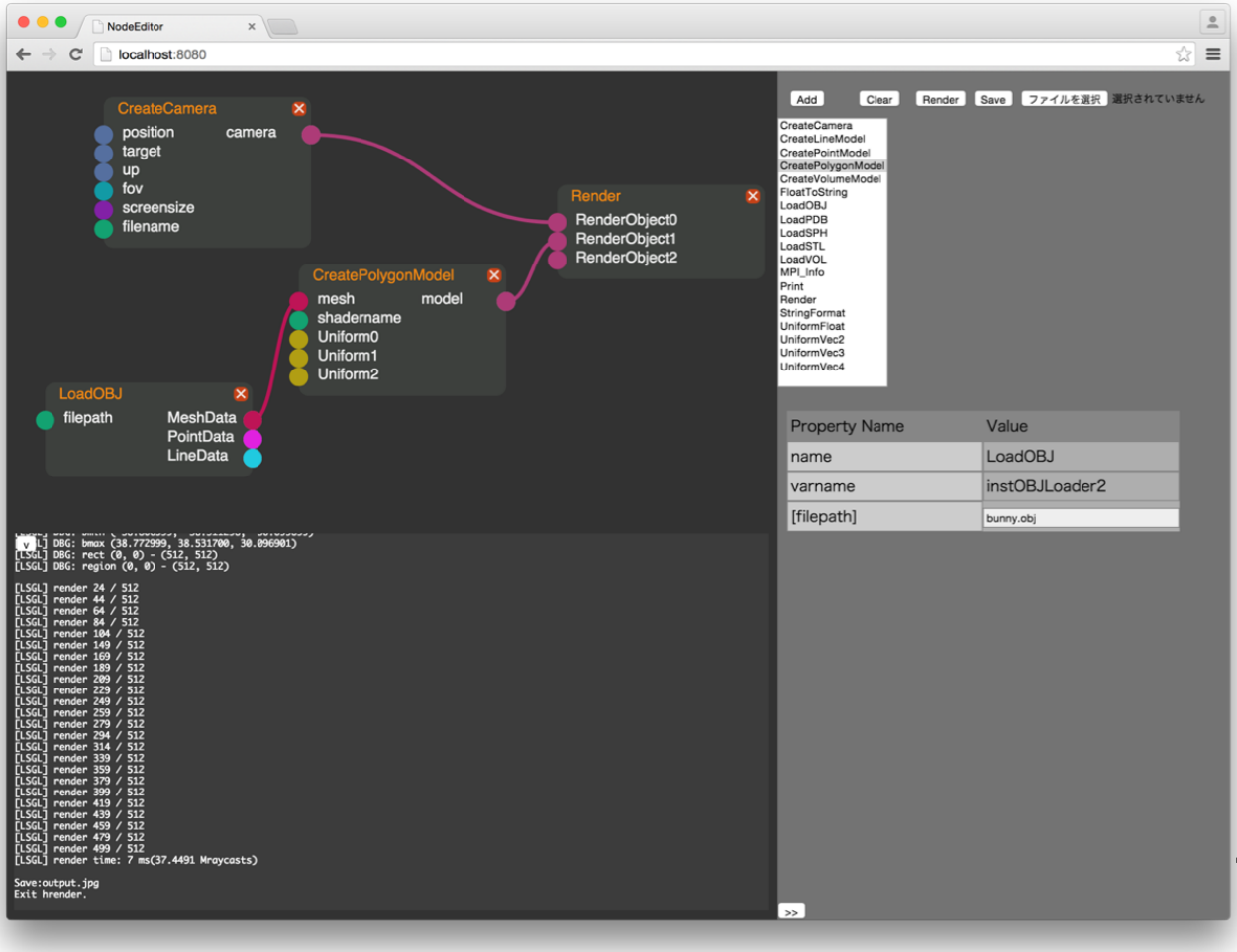

KVS (redis or mongoDB)
Image / SceneCommand(Lua) / Parameter(Json)
node-redis with hiredis(js)
standalone mode
websocket/REST(C++)
node.js server(js)
HIVE Renderer(C++)
socket.io(js)
GLES
Open GL
SURFACE(C++) [Raytracer]
Loader
Builder
Lua(C)
Scene file(Lua)
Browser UI(js)
UI JS
CSS
OpenMP
libcio, libcpm, ...
MPI

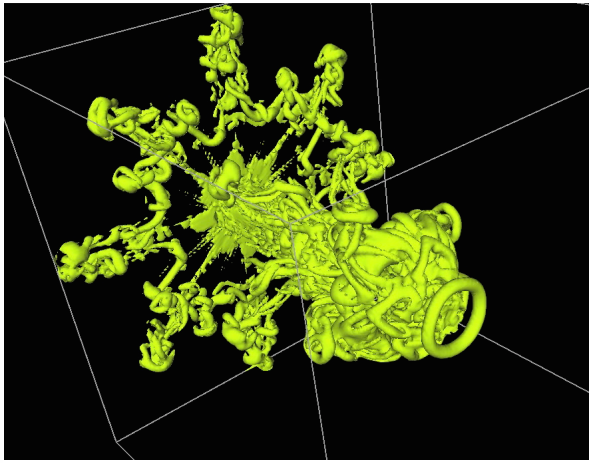
SURFACE

- **S**calable and **U**biquitous **R**endering **F**ramework for **A**dvanced **C**omputing **E**nvironment
  - Sort-Last Parallel Rendering
  - OpenGL ES 2.0 compatible API



Scene Node Editor



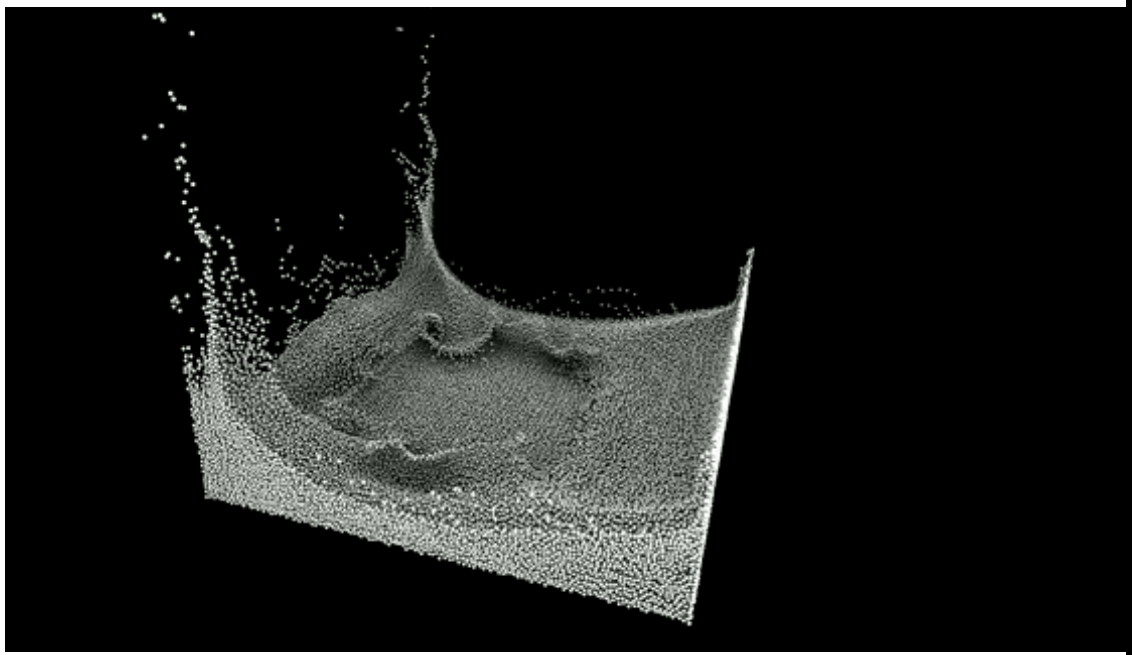


output.jpg

hrender

scene.scn

Examples



Particle tracing

Resolution of 4096 x 4096 pixels, Asian Dragon

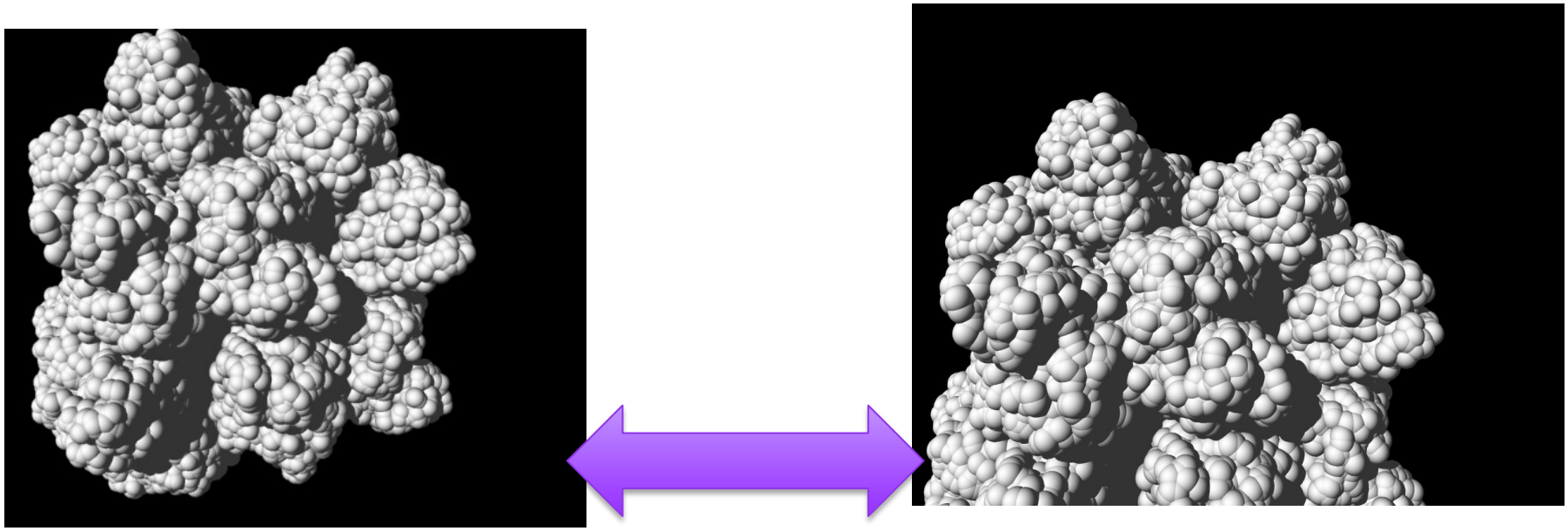
Off-line rendering of PDB data

Data :

http://www.rcsb.org/pdb/explore.do?structureId=1mt5

Only Atom, 1M

Renderig point primitives with Lambert shader and ray casting

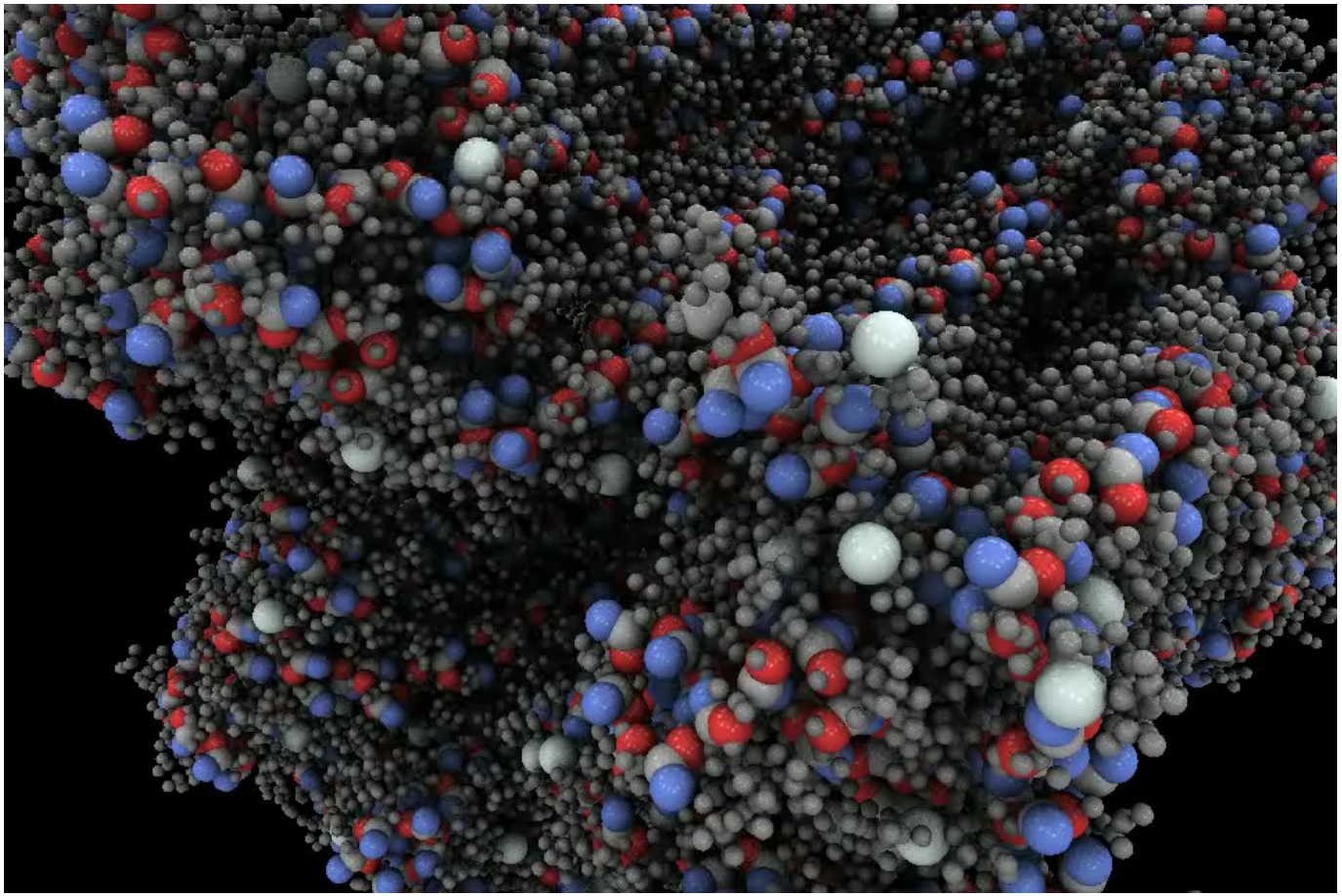


Result on Intel PC

Almost same image

Result on K

High Res. Rendering Image



2015-09-02　HIVE 講習会内容

http://www.aics.riken.jp/jp/outreach/aicssoft_training15-6.html

- HIVE について
  - システム構成について
  - HIVE のビルドとインストール
  - HIVE UI,SceneNodeEditor の起動と使いかた
- hrenderについて
  - hrender概要
  - データローダについて
  - フィルタ機能について
  - 可視化フロー例
  - MPI並列レンダリングについて
- シェーダについて
- SceneNodeEditorについて
  - SceneNodeEditorによる簡単な可視化例
  - NodeEditorで作成したシーンの hrenderでのレンダリング例
- HIVE UIについて
  - ボリュームレンダリングの伝達関数について
  - マルチカメラの設定について
  - HIVE UIでのムービー作成方法
  - タイムステップデータのムービー作成方法
  - HIVEUIへの独自シェーダ登録方法