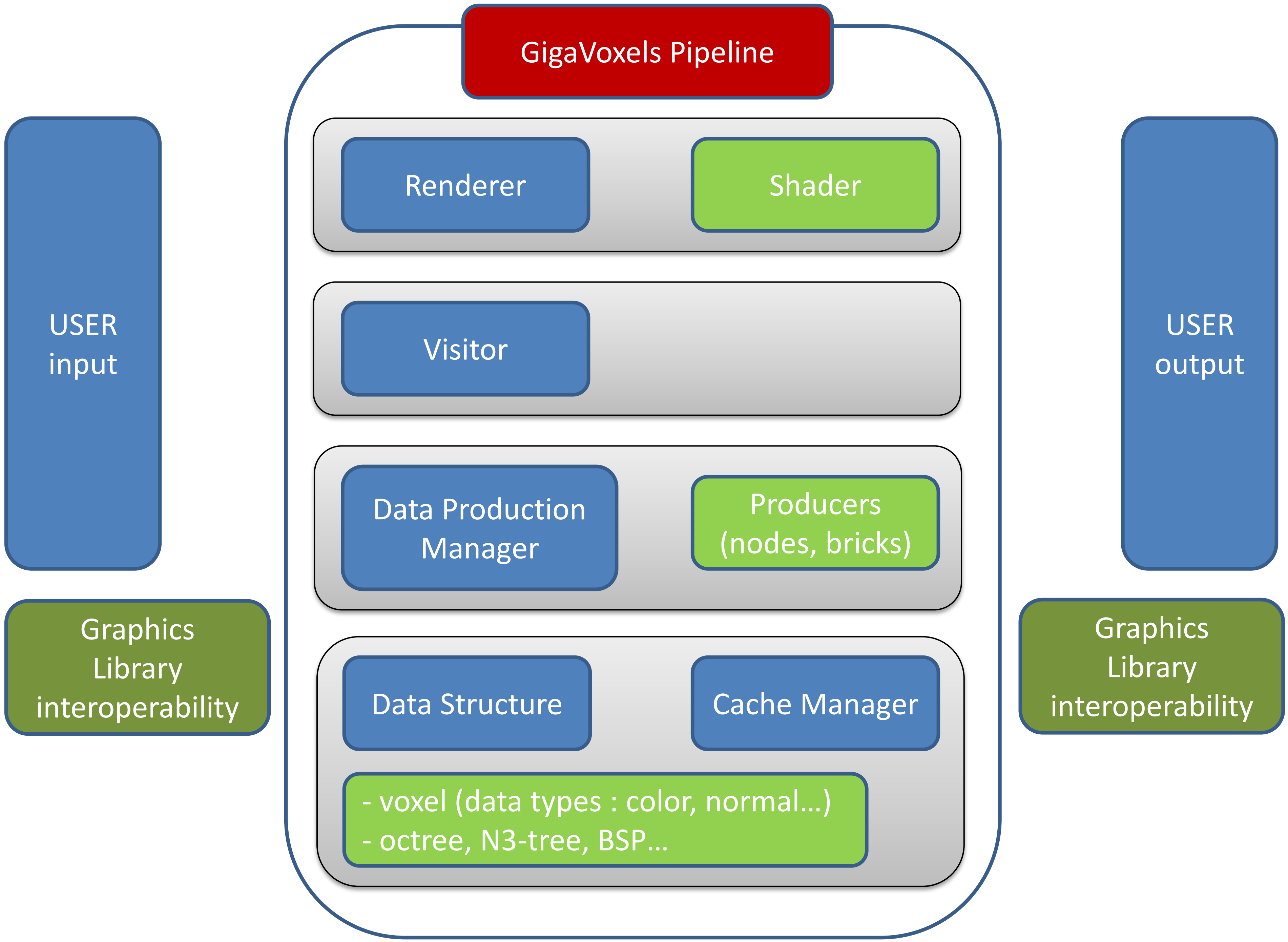
Key Features

- **Tree Data Management** (space partitioning) to store and organize data

  (octree or generalized N3-tree, + SDK example kd-tree)
- **Cache System on GPU** : LRU mechanismn (least recently used)

  (to get temporal coherency)
- **Data Production Management** : on host, GPU, or hybrid mode

  Goal : produced data are kept in cache on GPU
- **Visit algorithm** : traverse your data (loaded in cache) as could be done for rendering
- **Renderer** (hierarchical volume ray-casting, cone tracing, emission of requests, brick marching)

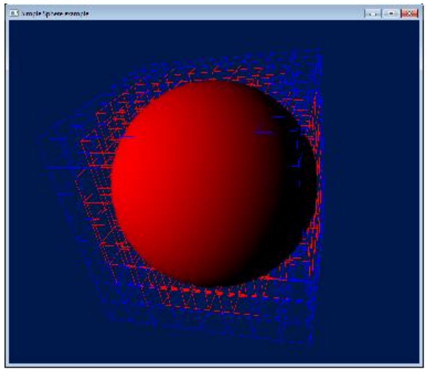

GigaVoxels Pipeline
USER input
Renderer
Shader
USER output
Visitor
Data Production Manager
Producers (nodes, bricks)
Graphics Library interoperability
Data Structure
Cache Manager
Graphics Library interoperability
- voxel (data types : color, normal...)
- octree, N3-tree, BSP...

Custom features

- **Voxel data type** : user can defined what's inside a voxel
  - list of types : uchar4 (ex: color), half4 (ex: normal)…
- **Producers** : fill the data structure
  - Node producer : fill spatial data structure
  - Brick producer : fill voxel data
  - either host or device, or a combination
  - based on user oracles
- **Shader** :
  - modify apperance by sampling data along rays

Data Structure [1/2]

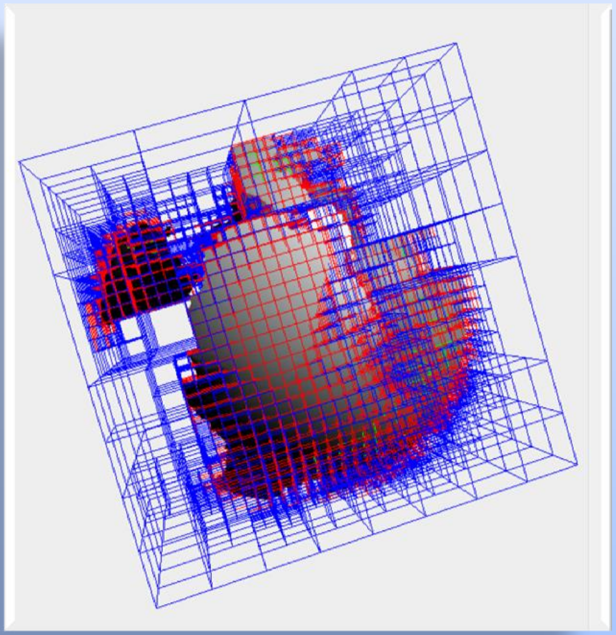
- **Exemple** : generate a sphere
  - user defined "voxel data type" : RGBA color + normal

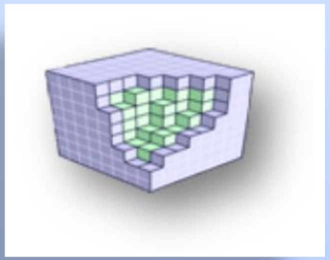


Generalized N3-tree of nodes (multi-scale)

Brick of voxels (user data stored in each node)

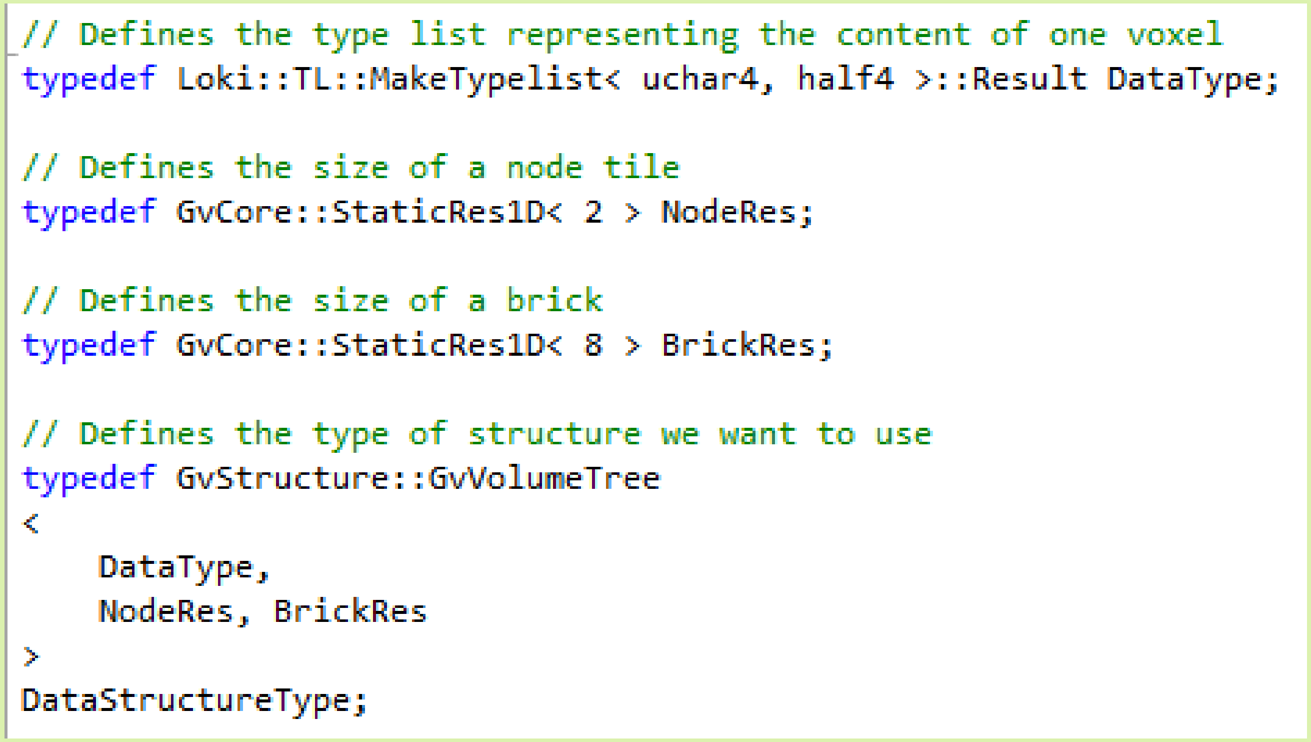
Data Structure





Data Structure [2/2]

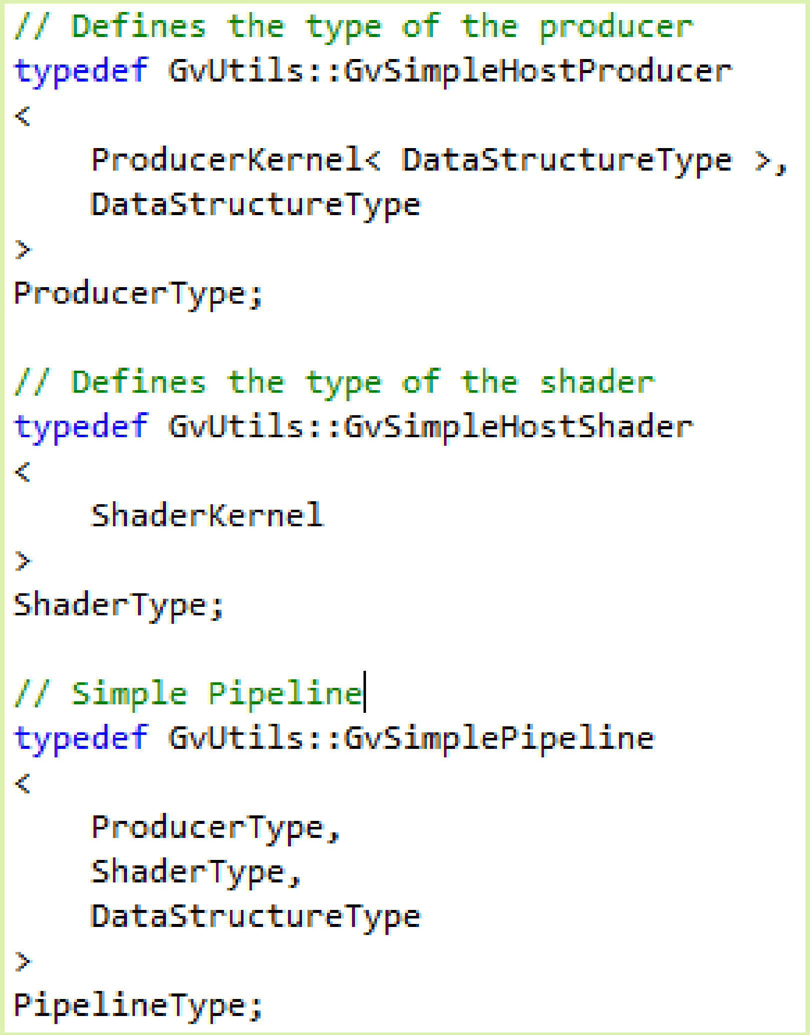
- **Voxel data type** : based on list of types [uchar4 (color), half4 (normal)]
- **Data Structure** : based on a generalized N3-tree
  - node resolution : nb of child in each node (for each dimension)
  - brick resolution : nb of voxels in each node (for each dimension)



```
// Defines the type list representing the content of one voxel
typedef Loki::TL::MakeTypelist< uchar4, half4 >::Result DataType;

// Defines the size of a node tile
typedef GvCore::StaticRes1D< 2 > NodeRes;

// Defines the size of a brick
typedef GvCore::StaticRes1D< 8 > BrickRes;

// Defines the type of structure we want to use
typedef GvStructure::GvVolumeTree
<
    DataType,
    NodeRes, BrickRes
>
DataStructureType;
```

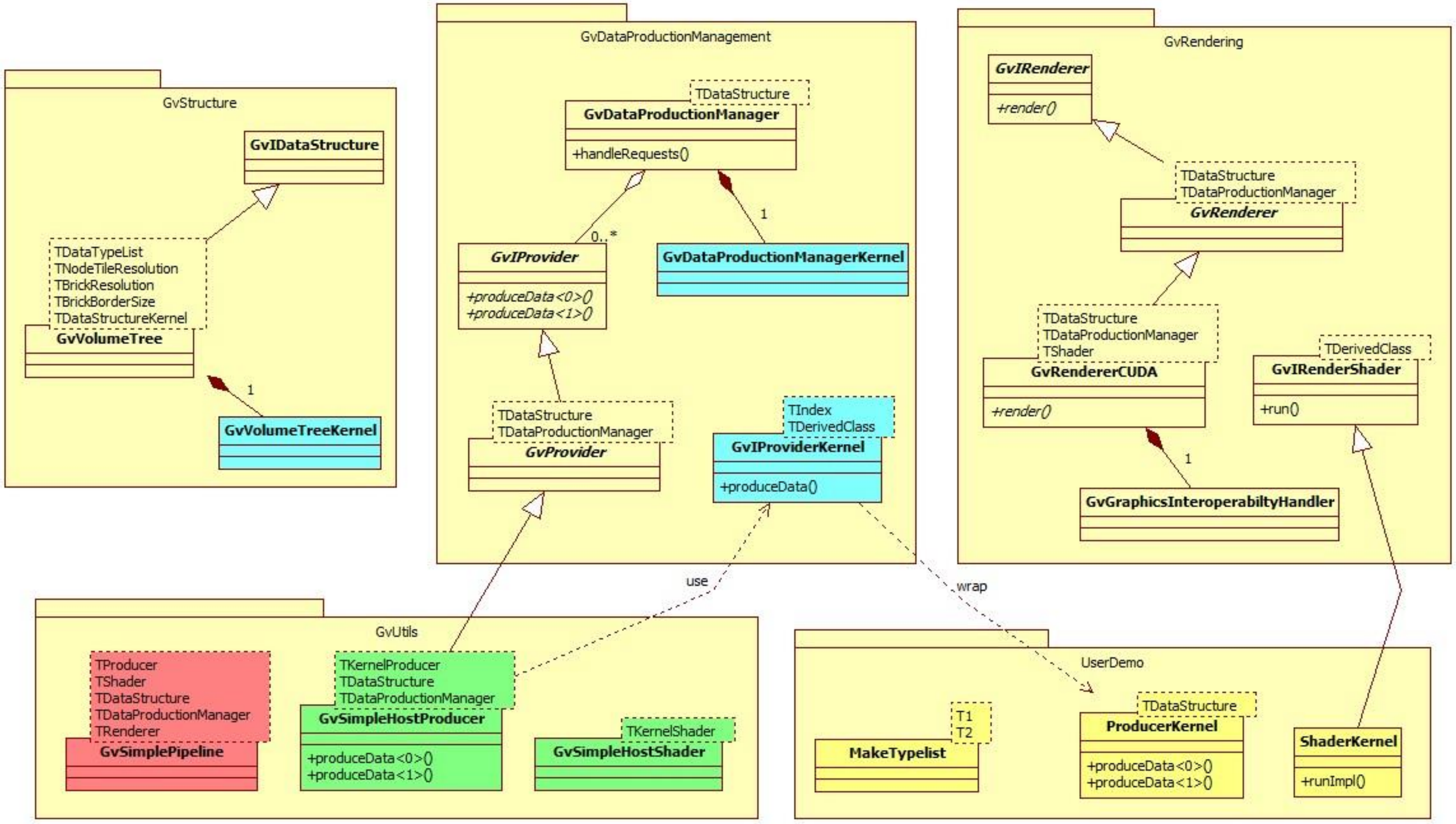
Data Structure [2/2]

- **GigaVoxels pipeline** : access all GigaVoxels objects (renderer, producers…)



```
// Defines the type of the producer
typedef GvUtils::GvSimpleHostProducer
<
    ProducerKernel< DataStructureType >,
    DataStructureType
>
ProducerType;

// Defines the type of the shader
typedef GvUtils::GvSimpleHostShader
<
    ShaderKernel
>
ShaderType;

// Simple Pipeline
typedef GvUtils::GvSimplePipeline
<
    ProducerType,
    ShaderType,
    DataStructureType
>
PipelineType;
```

Producers [2/2]


GvStructure
GvIDataStructure
TDataTypeList
TNodeTileResolution
TBrickResolution
TBrickBorderSize
TDataStructureKernel
GvVolumeTree
1
GvVolumeTreeKernel
GvDataProductionManagement
TDataStructure
GvDataProductionManager
+handleRequests()
0..*
1
GvIProvider
+produceData<0>()
+produceData<1>()
GvDataProductionManagerKernel
TDataStructure
TDataProductionManager
GvProvider
TIndex
TDerivedClass
GvIProviderKernel
+produceData()
GvRendering
GvIRenderer
+render()
TDataStructure
TDataProductionManager
GvRenderer
TDataStructure
TDataProductionManager
TShader
GvRendererCUDA
+render()
TDerivedClass
GvIRenderShader
+run()
1
GvGraphicsInteroperabiltyHandler
use
wrap
GvUtils
TProducer
TShader
TDataStructure
TDataProductionManager
TRenderer
GvSimplePipeline
TKernelProducer
TDataStructure
TDataProductionManager
GvSimpleHostProducer
+produceData<0>()
+produceData<1>()
TKernelShader
GvSimpleHostShader
UserDemo
T1
T2
MakeTypelist
TDataStructure
ProducerKernel
+produceData<0>()
+produceData<1>()
ShaderKernel
+runImpl()