**Design Goals of CGiS**

- **Unified language**
- Algorithm expressable in a single language
- Compiler takes care of distributing code, data
- **Portability**
- Across GPU generations / vendors
- **High-level programming languages**
  - Shading languages (Cg, HLSL, glslang)
  - Brook for GPUs
- **CGiS**

**GPGPU**

- Ongoing advances of GPU programmability
- Used for general purpose computations
- Good results, but tedious work
- High-level programming languages
  - Shading languages (Cg, HLSL, glslang)
  - Brook for GPUs
- **CGiS**

**Compiler Structure**

- CGiS Source
- Optimizing CGiS Compiler
- GPU Kernels
- SSE Kernels
- Glue Code
- GPU-RT
- SSE-RT
- Main Application

**Example**

```c
PROGRAM EarlyKillRayTriangleIntersection;

INTERFACE

typedef struct {
  float3 a, b, c;
  int id;
} tri_t;

typedef struct {
  float3 origin, direction;
  float near, far;
} raytype_t;

extern output int[2D] rayhits;
extern input tri_t[2D] t_list;

CODE

function intersection_triangle (output int tid, input raytype_t ray, input tri_t t) {
  // Perform Möller-Trumbore
  // intersection
  if (!intersect) {tid=-1; return;}
  tid = t.id;
}

function earlyKillIntersect (input raytype_t ray, output int tid, input tri_t[2D] triangleList) {
  do {
    intersection_triangle (tid, ray, t);
    if (tid != -1) break;
  }
}

CONTROL

forall (raytype_t r in raydata:2D; int tid in rayhits:2D) do {
  earlyKillIntersect (r, tid_t_list);
}
```

**Language Features**

- Parallel control structures
- Special vector operators
- Powerful data structures
- Multi-dimensional
- Array of structs
- User-defined
- User-specified hints
- Additional optimization
- Guides for code generation
- GPU/SIMD-CPU code generation
- Automatic SIMD optimization
- Direct performance comparison
- Easy debugging

**Differences to Existing Frameworks**

- Cg/HLSL/glslang:
  - Different application domain, implementation of shading kernels
- Brook for GPUs
  - Abstracts the GPU as a streaming coprocessor
  - No light-weight communication (occlusion query)
  - No inputs through vertex parameters
  - One kernel corresponds to one fragment program
  - Algorithms decomposed into GPU part (C++) and GPU parts (Cg)
- User has to decide which computations have to be done on the GPU

**Project Status**

- Language design and frontend finished
- Simple GPU backend working
- First few program analyses implemented
- Sponsored by DFG (Project WI 576/10)