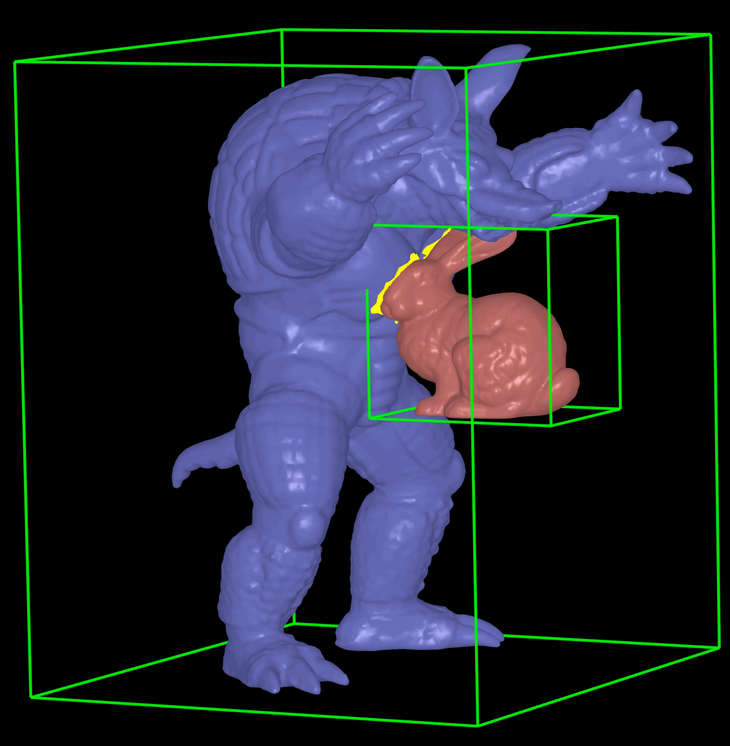
Using Metaprogrammed Functors to Implement Double-Dispatch for Collision Handling

Tiago H. C. Nobrega[1], Diego D. B. Carvalho[1], Aldo von Wangenheim[1]

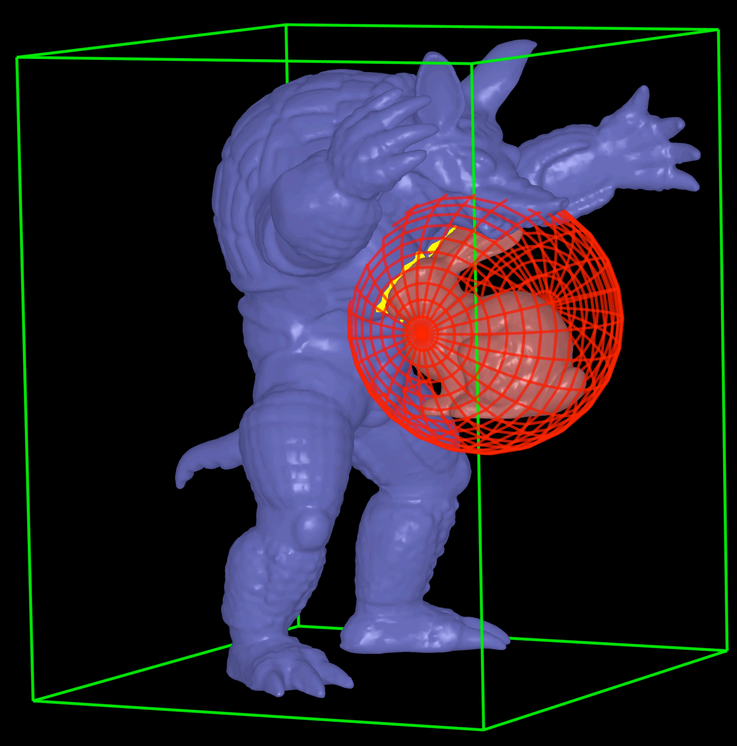
[1]LAPIX – Federal University of Santa Catarina – Florianópolis – Brazil

{tigarmo,diegodbc,awangenh}@inf.ufsc.br

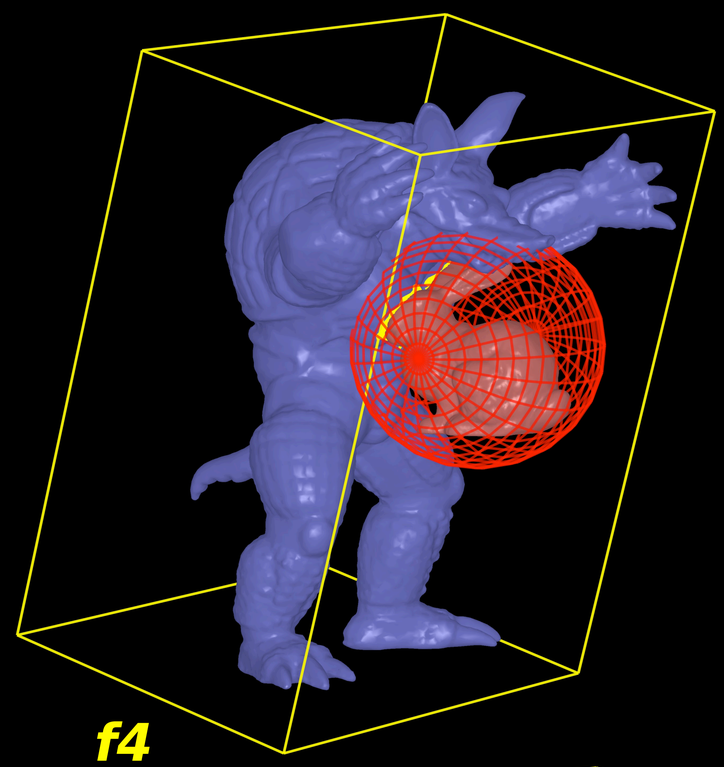
www.lapix.ufsc.br



f1

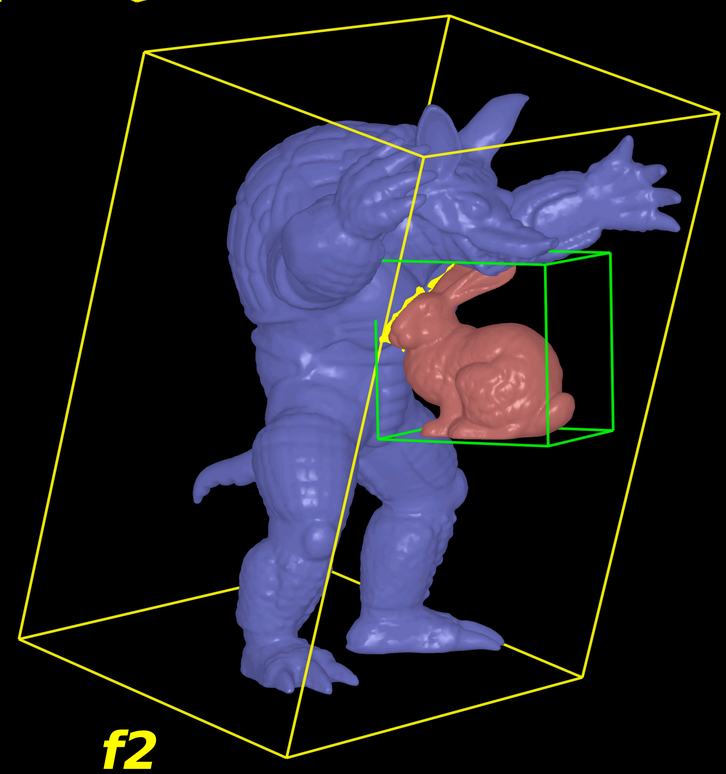


f3



f4

	AABBTree	OBBTree	SphereTree
AABBTree	*f1*	*f2*	*f3*
OBBTree	*f2*	—	*f4*
SphereTree	*f3*	*f4*	—



f2

Introduction

- Collision detection engines and frameworks are difficult to design and riddled with efficiency, modularity and cleanliness trade offs.
- The management of multiple object types colliding in a scene can be aided by the double dispatch mechanism.
- We show how both object-oriented and generic notions can be used together to implement efficient, clean double dispatch.

Bounding Hierarchies with Double-Dispatch and Metaprogramming

- Bounding trees [1] are commonly used in collision detection. It is natural and simple to have each bounding tree class inherit from a base class filled with common methods.
- Alexandrescu [2] shows how to use this kind of hierarchy to efficiently implement double dispatch with a matrix of function pointers.
- We expand this idea with *functors.*
- Some C++ metaprogramming tricks offload the chore of filling the matrix to the compiler.
- The functor templates can be specialized as needed —the compiler will find the right instantiation.

```
template<class TreeA, class TreeB>
struct CollisionFunctor{
  bool operator()(const TreeA&, const TreeB&);
};
```

Conclusions

- Initial benchmarks: at worst 15% overhead.
- The idea of generic functor matrices can also be applied to other dispatch cases.

References

[1] C. Ericson. Real-Time Collision Detection. Morgan Kaufmann, December 2004.

[2] A. Alexandrescu. Modern C++ design: generic programming and design patterns applied. Boston, MA, USA, 2001.