FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Diplomarbeitspräsentation

An Adaptive, Hybrid Data Structure for Sparse Volume Data on the GPU

Masterstudium:
Visual Computing

Matthias Labschütz

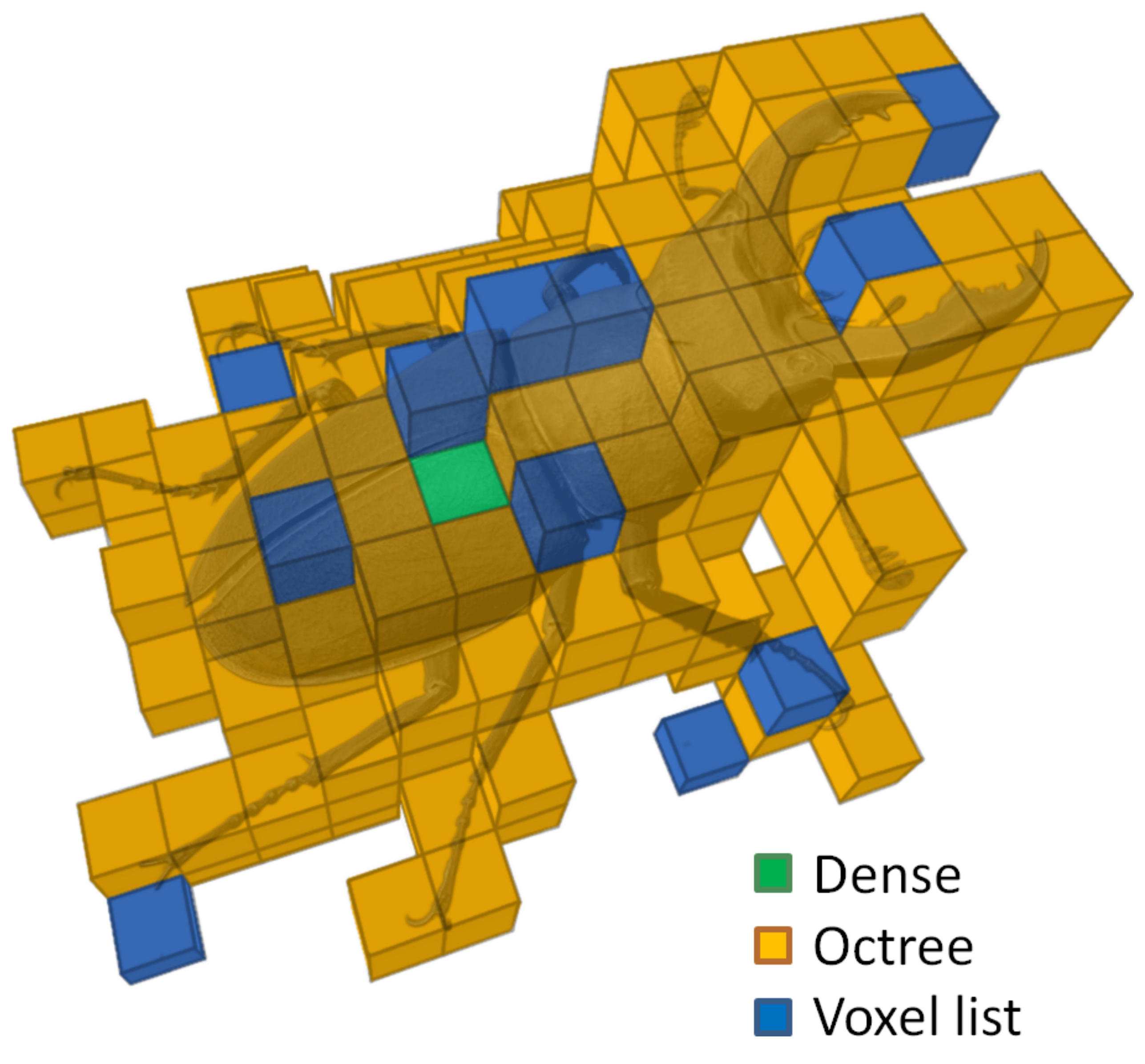
Technische Universität Wien
Institut für Informationssysteme
Arbeitsbereich: Computergraphik
BetreuerIn: Ao.Univ.-Prof. Dr. M. Eduard Gröller
Mitwirkung: Dipl.-Ing. Dr.techn. Peter Rautek

Motivation

Dealing with large, sparse, volume data on the GPU is a necessity in many applications such as volume rendering, processing or simulation. The limited memory budget of modern GPUs restricts users from uploading large volume data sets entirely. Fortunately, sparse data, i.e., data containing large empty regions, can be represented more efficiently compared to a common dense array. Our approach proposes an efficient representation for volume data that decreases the memory requirement for sparse data sets while maintaining high access performance. The concept of the JiTTree was presented in the journal paper *JiTTree: A Just-in-Time Compiled Sparse GPU Volume Data Structure* [1], and was presented at the IEEE Visualization Conference 2015, Chicago, USA.

Hybrid Volume Bricking

We extend the concept of traditional volume bricking to support four different brick types with varying properties. We implement **dense volume**, **octree**, **voxel list** and **empty bricks**.



This makes it possible to better fit most data sets, which reduces the memory requirement of sparse volume data on the GPU. Our approach can be extended to support additional brick types. The drawback of this hybrid representation is that it reduces the access performance. To improve the access performance we introduce JiTTree.

Results & Future Work

We additionally compare **15 test data sets**. Hybrid bricking **reduces the memory requirement** significantly for all of them. JiTTree increases the access performance for certain access patters such as the stencil access in mean filtering. For mean filtering, we achieve an average **performance gain of 1.29**.

We have presented two promising ideas. The first was the introduction of a hybrid data structure, which was deduced from a modular data structure. We believe that a more general modular data structure could eventually lead to a variety of high performing data structures which are not known or widely used today. The second concept was to use data dependent code specialization on the GPU. We suspect that other GPU based applications could benefit from data dependent code specialization. A further investigation could lead to new insight about the benefits and drawbacks in a more general setting.

JiTTree

JiTTree replaces hybrid bricking with a kd tree of bricks. A data-aware compilation step transforms general kd tree traversal code into specialized OpenCL kernel code. Since data sets are typically loaded at the run-time of the program and we construct JiTTree afterwards, we call our data structure just-in-time compiled tree.

Traditional kd tree traversal:

```
int get(int[] pos, Node node)
{
  if (node.type != LEAF) { // recursion
    if (pos[node.axis] < node.key)
      get(pos, node.leftChild);
    else
      get(pos, node.rightChild);
  } // type checks for every leaf node
  else if (node.type == OCTREE)
    return getOctree(...);
  else if (node.type == VLIST)
    return getVList(...);
  else if (node.type == DENSE)
    return getDense(...);
  else return EMPTY;
}
```

⇨

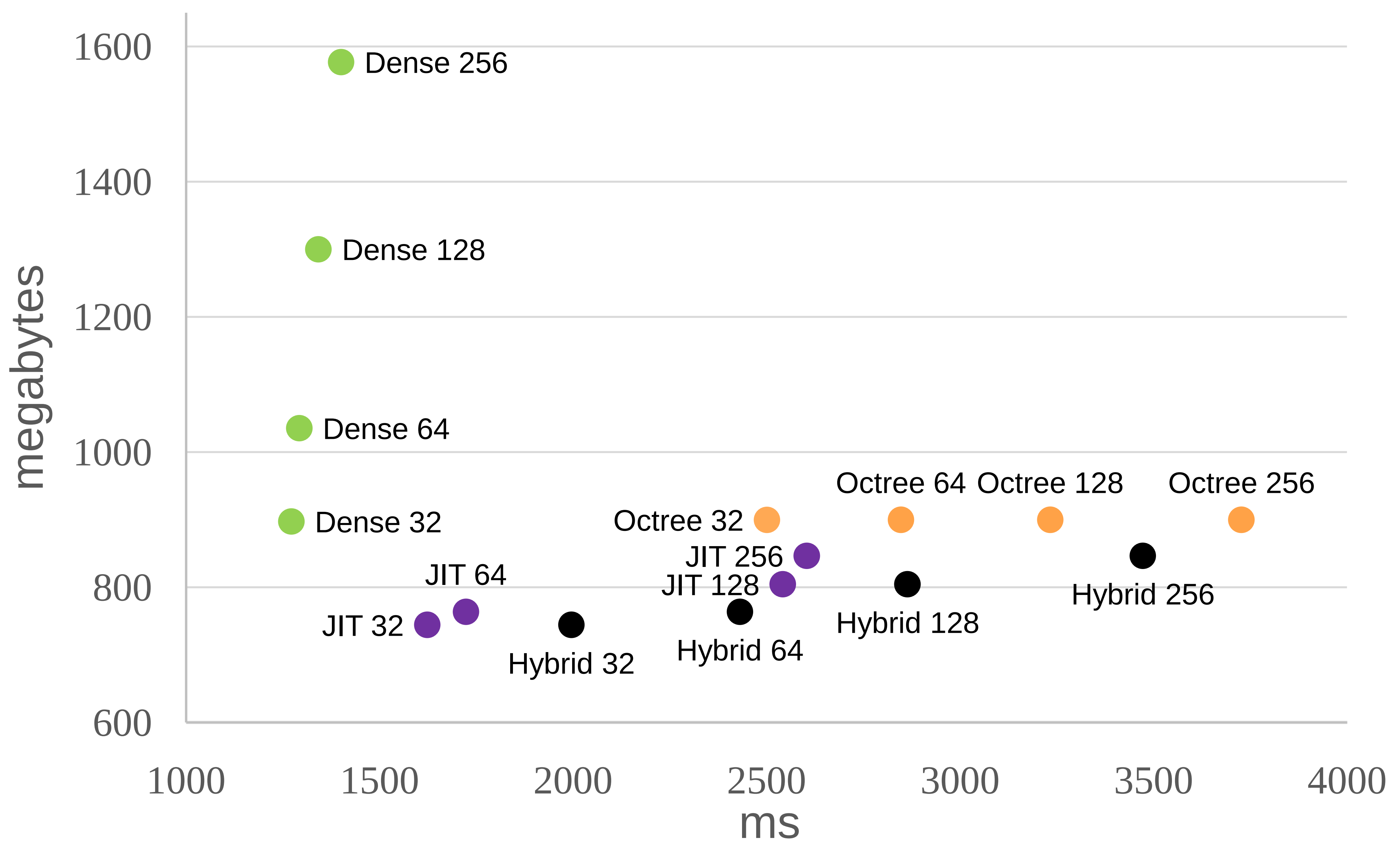
JiTTree traversal:

```
int get(int x, int y, int z)
{
  if (x < 192) {
    return getOctree(...);
  } else {
    if (z < 384) {
      if (x < 384)
        return getVList(...);
    } else {
      if (y >= 192)
        return getDense(...);
    }
  }
  return EMPTY;
}
```

Global memory fetches into the node data structure are transformed to instruction fetches. JiTTree directly writes the boundary values of the kd tree to their corresponding conditions and it is not necessary to keep track of the splitting axis direction. The kd tree of bricks implicitly handles type routing. Each node of the kd tree can directly execute its type specific traversal code.

Memory Consumption & Performance

The Figure shows the memory and performance characteristics of mean filtering a data set with $1024 \times 1024 \times 1024$ voxels. Traditional **dense bricking** with varying brick size is compared to **octree bricking** and **hybrid bricking**. The **JiTTree** results provide a higher performance with the lowest memory requirement.



References

[1] Labschütz, M., Bruckner, S., Gröller, M. E., Hadwiger, M., and Rautek, P. (2016). JiTTree: A just-in-time compiled sparse GPU volume data structure. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):1025–1034.

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Kontakt: labschuetz@gmail.com