AUTOMATYKA • 2011 • Tom 15 • Zeszyt 3

Bartosz Waresiak*, Paweł Skrzyński*

Using Quad Tree as Data Storage for a Terrain Representation and a Core for a Path Finding Algorithm

1. Introduction

Quad tree data structure was introduced by Raphael Finkel and J.L. Bentley in 1974 [2]. In their article they present algorithms used to build and optimize a quad tree and discuss the complexity of both: inserting and retrieving operations. Nowadays, many solutions in various research areas are based on their work, some of which are presented below.

One of the examples of quad tree usage is image data mining that uses quad trees as a part of a retrieval engine that can efficiently match remotely sensed imagery. In the article titled *Searching satellite imagery with integrated measures* [7] authors describe a system, which uses hierarchical representation for computing spectral and spatial attributes of images. Due to a couple quad tree with second-level spatial autocorrelation it is possible to take advantage of both (spectrum and space) features of an image in an efficient way. Another example related to image processing is the usage of quad trees in the visualization of stacked multi-resolution images. Interactive visualization of multi-resolution image stacks in 3D [9] presents an approach in which each image consists of a hierarchical image pyramid made from small image tiles. A quad tree is adapted there to organize tiles and represent multiresolution images. Chen, Szczerba and Uhran in Planning conditional shortest paths through an unknown environment: a framed-quad tree approach [1] write about searching patches in an obstacle-scattered environment, where information about obstacles are provided by external sensors and a path is computed "on the fly". Distance is calculated with the use of a circular path-planning wave. The authors proposed a method, which uses a framed-quad tree technique to represent the environment. This technique combines accuracy of grid-based path planning and the efficiency of a quad tree. The dynamic partitioning of the system and the later allocation of these partitions to the incoming tasks that uses quad-tree based processor-allocation algorithms is described in article written in 2005 [3].

* AGH University of Science and Technology, Faculty of Electrical Engineering, Automatics, Computer Science and Electronic, Department of Electronics, al. A. Mickiewicza 30, 30-059 Krakow, Poland

A quad tree is used by the authors in order to represent information about the allocation of sub-meshes in a given mesh-connected system. As a result internal fragmentation is minimized and the largest possible free subsystem is maintained. Optimization is another area, where quad trees are being used. An article titled *A primogenitary linked quad tree data structure and its application to discrete multiple criteria optimization* [8] presents an extension of traditional quad tree data structure, which leads to savings in memory or storage space as well as in execution time. Authors make a wide comparison between those two data structures and show important differences. Besides those mentioned above, more applications of this data structure exist, and new ones are being researched every year.

2. System concept

As described in the previous section, quad trees can be successfully used to solve many types of problems related to computer science. In further sections we focus on using a single, universal quad tree as both: data storage for a terrain data and a core for a path-finding algorithm.

Based on this concept, various software system might be developed, the use of which include, but is not limited to: supporting navigation of the ships, calculating possible routes in areas with rough, mountain terrain or being a part of Geographic Information System.

Using quad trees for storing and displaying data was described in a recent article titled *Modeling of Earth's Gravity Fields Visualization Based on Quad Tree* [6]. In the article, the authors store the earth's gravity field data in a quad tree, and then visualize it using a Triangle Fan approach to render each nodes. The quad tree hierarchical structure is used to include level of detail (LOD) which enables multi-resolution rendering that improves overall display performance. Both node rendering and LOD application is described in more detail in further sections of this article.

In an article titled *Multi-resolution Path Planning for Mobile Robots* [5] the authors describe how to use quad tree nodes to perform A* [4] path-finding for mobile robots. Staged (hierarchical) path planning based on quad tree multi-resolution representation is used – this approach appears to be more efficient compared to a typical A* search.

2.1. Quad tree structure

Based on the above applications, the idea of using a single quad tree used to perform both tasks has emerged. In the implementation proposed by the authors, path-finding is tested on an artificially created two-dimensional world that consists of a water area and a single island in the middle. The land is considered to be an impassable terrain, and the result of path-find operation is a possible route for a ship that has to avoid the island.

A quad tree structure consists of many node elements. Each of them stores coordinates and height of map space that is represented by quad tree. Besides terrain data, more utility

parameters and parameters related to path-finding are included in every node. They are further described in the implementation section of this article.

The construction of the tree is founded on an algorithm that uses the recursive decomposition of a two-dimensional terrain data. An example of an algorithm is presented in Listing 1.

```
RECURSIVE_DECOMPOSITION(Node):
    IF( Node.shouldDecompose() ) //STOP condition
        Node.decompose();
        FOREACH( quarter : node.quaters )
            RECURSIVE_DECOMPOSITION(quarter);
    ELSE
        RETURN;
```

Listing 1. quad tree creation algorithm in pseudo code

For the purpose of path-finding in this article, *shouldDecompose()* function returns *FALSE*, if either node contains no more obstacles (there is no land terrain within a node boundary), or size of the node reached minimal acceptable size. The first condition ensures that once there is no need for a decomposition, a leaf node will optimally represent a big part of the terrain. The second condition guarantees that the tree will not grow infinitely.

It is important to mention that while the first condition improves the efficiency of path finding, it is also valid for the purpose of storing terrain data based on terrain constraints. (Because a terrain with no obstacles represents water, and the height of such a'terrain' is constant and equal to 0.) Hence, it is not necessary to further divide such nodes for representational purposes because no additional information about terrain can be obtained while dividing up a 'water node'.

Contrary to water, 'land nodes' have to be divided up until second part of the algorithm's stop condition (minimal size) which finishes the decomposition process. It is because each time a land node is divided, new information about terrain height for newly accessed points may be obtained.

The resulting tree structure strongly depends on terrain characteristics. If the terrain contains many open spaces, the number of nodes in a tree will be significantly smaller than if the terrain consisted of many scattered obstacles.

Figure 1 presents two examples of how the shape of the terrain affects its quad tree representation. The white area is considered as passable, while black nodes represent obstacles.

Map 'A' consists of big open spaces and the resulting quad tree based on this map contains some shallow branches with big leaf nodes. This is advantageous because bigger leaf nodes on the tree result in fewer nodes that have to be traversed while searching for a path. In the best-case scenario, both start and destination belong to the very same node – the resulting path is then just a straight line between these points.

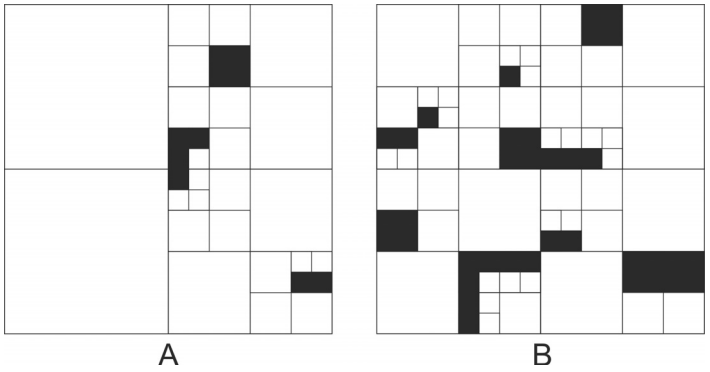

Fig. 1. Examples of map decomposition using a quad tree

Map ‘B’ contains various, scattered obstacles. The resulting tree is bigger and more complex than the previous one, but the advantages of recursive space decomposition are still apparent – space is divided up only if it is necessary.

2.2. Level od Detail

As already mentioned, the hierarchical structure of quad trees allows for a relatively simple implementation of Level of Detail (LoD) functionality. This functionality improves the performance of the image rendering, by reducing the amount of displayed data when possible. Omitted data must have a minimal influence on overall display precision to make sure that no important information is lost. If the method is properly implemented, then despite the fact that some details are lost in the process, the result is still relatively detailed.

Figures 2 and 3 present examples of two situations: when LoD can and cannot be successfully applied.

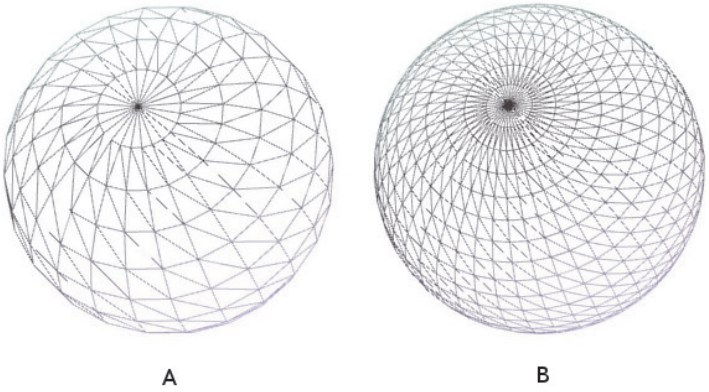

Fig. 2. Example od LoD for distant objects

Sphere ‘B’ from Figure 2 consists of a big amount of triangles, while sphere ‘A’ is the same sphere, but with a reduced number of triangles. With such a presentation, sphere ‘B’ can be replaced by sphere ‘A’ with a minimal loss of details. An amount of triangles used to draw the shape of the sphere will be significantly smaller.

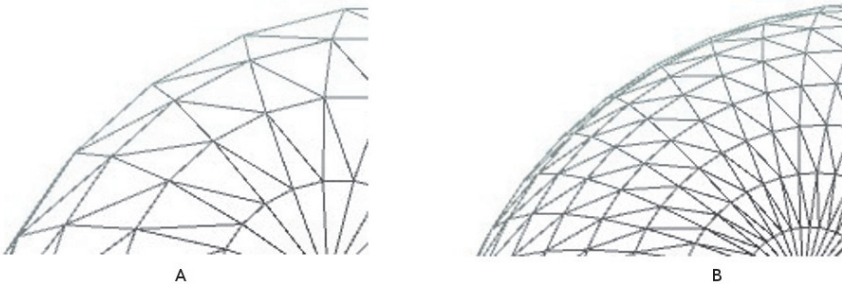

Fig. 3. Example of LoD for objects observed in detail

Figure 3 presents the same spheres that were shown in Figure 2, but after zooming in. In this case, it is clearly visible that reducing the amount of triangles in the case of sphere ‘A’ results in an image losing a significant amount of details compared to sphere ‘B’.

In the case of quad tree, which was created for the purpose of this article, the amount of presented details depends on nodes that are rendered on the screen. If a user observes a map from a distance nodes that are shallowly placed in the tree may be used for display. Once a user zooms-in, it may be required to render images in more detail by using leaf nodes to display terrain. Figure 4 presents an example of quad tree structure with selected nodes being the basis for rendering. Selection assumes LoD enabled, with a chosen detail level that corresponds to a maximum tree traversal depth of 2.

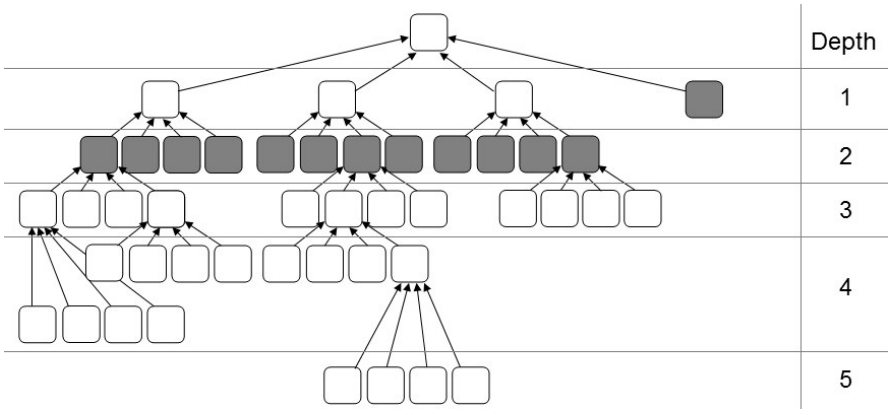

Fig. 4. Quad tree structure with example of nodes chosen while rendering terrain with LoD enabled

2.3. Path-finding

For the purpose of this article, path-finding is based on an A* algorithm, and a search is performed on a graph created only from linked quad tree leaf nodes. Figure 5 presents an example of the same quad tree structure as in Figure 4, but with selected nodes being the leaf nodes that are used for the path searching.

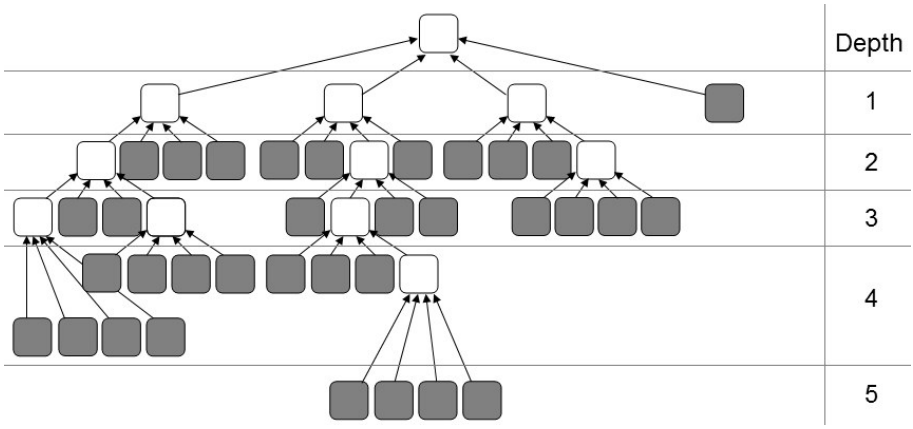

Fig. 5. Example of a quad tree structure with marked nodes being the leaf nodes

Further optimization involves removing from the path-finding graph nodes that represent impassable terrain, which are used only for rendering purposes.

A* algorithm uses evaluation function $f(n)$ that represents the optimal cost of a path traversal between two selected points, and an implementation discussed in this article is defined as:

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the cost of an optimal path from starting point to the current point, and $h(n)$ is the heuristic prediction of the traversal cost from the current point to the destination point. Function $g(n)$ is defined as:

$$g(n) = \sum_{z,z'} c(z,z'), \quad z,z' \in A \ \ and \ z \ follows \ z'$$

where set A consists of all the nodes that belong to the already calculated path, and $c(z, z')$ is cost of traversing from the a z node to the z' node (which precedes the z node), and is equal to the Euclide and the distance between nodes' centers.

$$c(z) = \sqrt{(x_z - x_{z'})^2 + (y_z - y_{z'})^2}$$

The heuristic function $h(n)$ is also based on the Euclidean metric and is equal to the distance between the canter of node n and the destination point.

$$h(n) = \sqrt{(x_n - x_{dst})^2 + (y_n - y_{dst})^2}$$

A* algorithms, are a family of minimal cost graph search algorithms, which optimality was sucesfully proved [4]. Usage of heuristic estimate $h(n)$ greatly improves computational complexity, but overall performance of path finding depends on both terrain characteristics (as it was mentioned in section 2.1) and distance between the source and the destination points.

3. Implementation

The implemented quad tree consists of the Nodes described in listing 2.

```
enum NODE_TYPE {
        WATER, LAND
};

class Node {
public:
//tree-structure - - - - - - - - - - - - - - -
        Node* parent;
        Node* children[4];

//map data - - - - - - - - - - - - - - -
        //heights
        double heightmap[9];

        //coordinates
        double upperLeft[2];
        double lowerRight[2];
        double center[2];

        NODE_TYPE type;

//pathfind-related parameters - - - - - - - - - - - - - - -

        std::vector<Node*> neighbours;

        bool traversed;
}
```

Listing 2. Node definition in C++

Node elements responsible for creating and maintaining tree structures are pointers to parent and four children nodes.

The part responsible for storing terrain information used to render the shape of a terrain consists of the coordinates of the upper left and lower right bounds of a node, coordinates of a center point and an array with terrain height data. Storing coordinates at the center of the node may seem redundant since its coordination may be easily calculated based on coordinates of upper-left and lower-right corners, but this information is stored for the purpose of performance optimization. A node is drawn using a Triangle Fan approach, and information about the center coordinates of each node is accessed every frame for each node that is displayed. Image 'A' on a Figure 6 present node coordinates – point *x* marks coordinates of the upper-left corner, point y coordinated of centre, and point z coordinates of the lower-right corner of a node. Part 'B' of this Figure shows nine points for which height data is stored within a node, and for which triangles are drawn inside the Triangle Fan loop. The numbers on an image correspond to point index numbers in the height-map array.

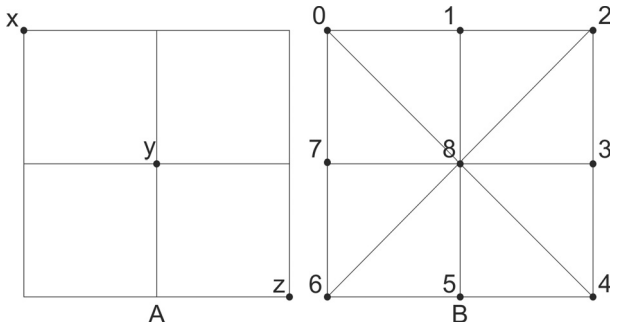

Fig. 6. Terrain information stored in a quad tree node

"Type" parameter describes the kind of terrain encapsulated by this node. For the purpose of this article, the testing world consists of two types of terrain: water and land. Water will be rendered with a light-grey color, land with a dark-grey color.

Each of the leaf nodes used in the path-finding algorithm contain an array of neighbor leaf nodes. Created by this relation graph is base for a path find algorithm. Each of the nodes contain a 'traversed' flag that is used during the execution of an algorithm to mark which nodes were already traversed. Figure 7 presents screenshot from running the application, with direct neighbors of a randomly selected node (marked with a white circle) being drawn with a dark-grey color.

Example of an LoD implementation is shown on Figure 8 and 9.

Figure 8 displays Level of a Detail turned off, which results in all of the leaf nodes being drawn. Overall count of rendered triangles in this case is close to 20 000. Figure 9 shows the result of turning on LoD. In this case only approximately 2 000 triangles are being displayed.

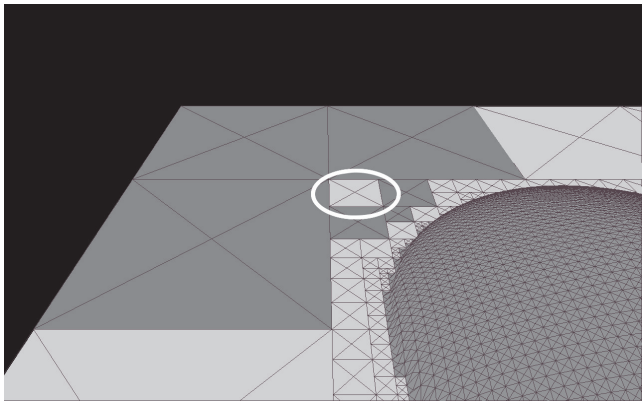

Fig. 7. Visualisation of node neighbours

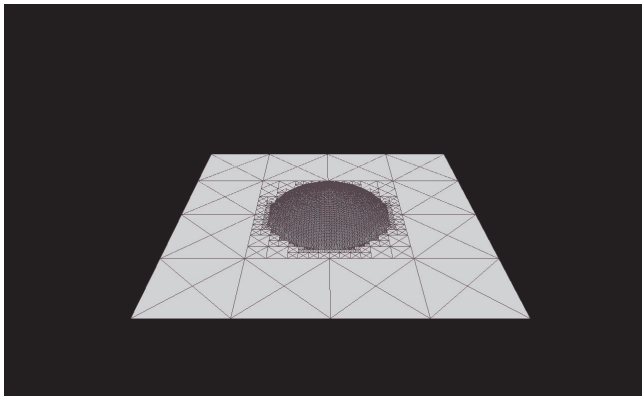

Fig. 8. Rendered terrain (water + island) without the LoD

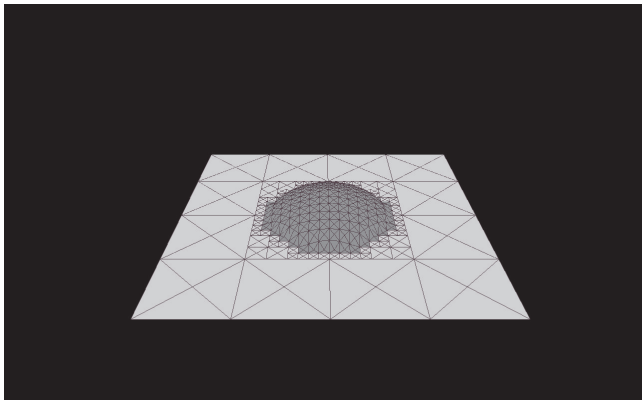

Fig. 9. Rendered terrain with LoD enabled

Some examples of path obtained in the path-finding process is shown on Figure 10 and 11.

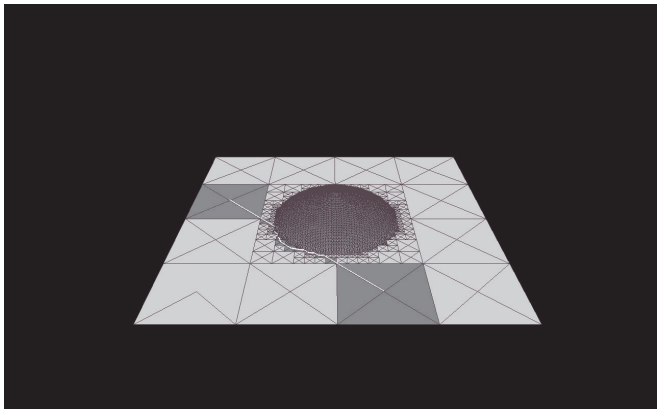

Fig. 10. Example of a path found on the water (avoiding impassable terrain)

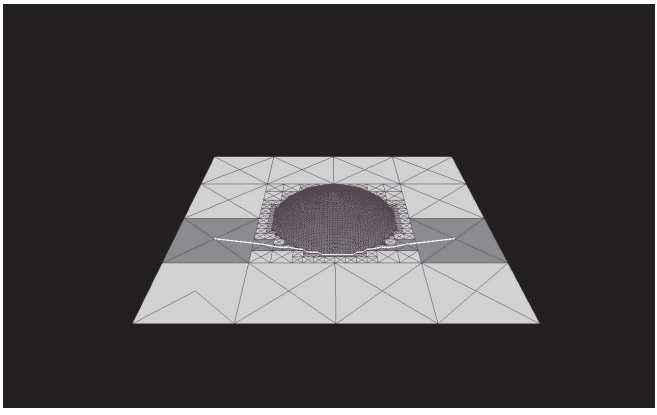

Fig. 11. Another example of path-finding

4. Conclusions

As we proved in this paper it is possible to use quad trees for data storage for terrain representation and a core for path-finding at the same time, in an efficient way. Quad tree search is really efficient which is achieved by the way a map is represented. Usage of LoD further improves efficiency.

Future work and possible research paths include work on memory usage optimization by decreasing the amount of data that is required to be stored in a node, supporting more kinds of terrain and adding the possibility to specify different terrain traverse costs and

cost-vectors that would take into account hill/mountain slopes during the path-finding's cost estimation. Further work on the improvement of the path-finding by taking advantage of quad tree hierarchical structures is also worth investigating. The idea of using NASA's Earth height-map data to create a representation of real scenarios or creating Geographic Information Systems based on this implementation are promising paths of research as well.

References

[1] Chen D., Szczerba R., Uhran J., Schnabl H., *Planning Conditional Shortest Paths Through an Unknown Environment: A Framed-Quadtree*. IEEE, 1995.

[2] Finkel R.A., Bentley J.L., *Quad Trees – A Data Structure for Retrieval on Composite Keys*. Acta Informatica 4, Springer-Verlag, 1974, 1–9.

[3] Gabriani G., Mulkar T., Szuba T., *A quad-tree based algorithm for processor allocation in 2D mesh-connected multicomputers.* Computer Standards and Interfaces, vol. 27, 2005, 133–147.

[4] Hart P., *A Formal Basis for the Heuristic Determination.* IEEE Transactions of Systems Science and Cybernetics, vol. SSC-4, No. 2, July 1968.

[5] Kambhampati S., Davis L., *Multiresolution Path Planning for Mobile Robots*. IEEE Journal of Robotics and Automation, vol. RA-2, No. 3, September 1986, 135–145.

[6] Luo Z. Li Z., Zhong B., *Modeling of Earth's Gravity Fields Visualization Based on Quad Tree*. Geo-spatial Information Science, vol. 13, issue 3, September 2010, 216–220.

[7] Samal A., Bhatia S., Vadlamani P., Marx D., *Searching satellite imagery with integrated measures.* Pattern Recognition, vol. 42, 2009, 2502–2513.

[8] Sun M., *A primogenitary linked quad tree data structure and its application to discrete multiple criteria optimization*. Ann Oper Res, vol. 147, 2006, 87–107.

[9] Trotts I., Mikula S., Jones E.G., *Interactive visualization of multiresolution image stacks in 3D.* Neuroimage, vol. 35, 2007, 1038–1043.