Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

qwViz: Visualisation of quantum walks on graphs ☆

Scott D. Berry [a], *, Paul Bourke [b], Jingbo B. Wang [a]

[a] *School of Physics, The University of Western Australia, Crawley, WA 6009, Australia*
[b] *iVEC at The University of Western Australia, Crawley, WA 6009, Australia*

ARTICLE INFO

Article history:
Received 15 March 2011
Received in revised form 6 June 2011
Accepted 7 June 2011
Available online xxxx

Keywords:
Quantum walk
Visualisation
OpenGL
Graph drawing
3D graphics

ABSTRACT

qwViz is a software package for interactive visualisation of the time-evolution of quantum walks on arbitrarily complex graphs. The package is written in C and uses OpenGL to generate graphics in real-time. The *qwViz* package can be used to directly simulate discrete-time quantum walks on undirected graphs when provided with the adjacency matrix of the graph. For more detailed studies, *qwViz* can also be used to visualise externally generated quantum walk data written in an XML-based file format (QWML). Various aspects of the visualisation can be customised and manipulated in real-time, allowing quantum walk dynamics to be probed at various length and time scales.

Program summary

Program title: qwViz
Catalogue identifier: AEJN_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEJN_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: GNU General Public Licence version 3
No. of lines in distributed program, including test data, etc.: 42 881
No. of bytes in distributed program, including test data, etc.: 97 152
Distribution format: tar.gz
Programming language: C
Computer: 32-bit and 64-bit workstation
Operating system: Linux, Mac OS X 10.5 (and later)
RAM: Depends on size of graph, typically less than 50 MB
Classification: 4.15, 14
External routines: OpenGL, GLUT, Graphviz [1]
Nature of problem: Simulation and visualisation of quantum walks on arbitrarily complex undirected graphs.
Solution method: The program uses OpenGL to produce 3D visualisations of time-dependent probability distributions arising from quantum walks on graphs. Graph layouts are automatically generated using Graphviz libraries.
Restrictions: Graph layouts are two-dimensional, with the third spatial dimension being used to represent the probability of finding the quantum walker at a certain location.
Unusual features: The software can be used in active or dual-stereo modes for 3D visualisation of quantum walks. Images and image sequences for movies can be exported in TIFF and TGA formats.
Additional comments: Examples of various input files and an XML schema are provided. Source codes written in C and Fortran are also supplied for generating QWML files from external quantum walk simulations.
Running time: Computing quantum walk data and graph layout for a 500-step quantum walk on a fifth-generation Sierpinski gasket (366 vertices) took less than 2 seconds on a 2.53 GHz Intel Core 2 Duo processor with 4 GB of RAM and 3 MB L2 shared cache under Mac OS X 10.6.6. The same simulation for a hyper-branched fractal with 1331 vertices took less than 25 seconds. GNU C compiler with optimisation option -O2 was used for these tests. Once data has been computed, the interactive visualisation can be manipulated in real-time.

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding author.
E-mail address: scottdberry@gmail.com (S.D. Berry).

0010-4655/$ – see front matter © 2011 Published by Elsevier B.V.
doi:10.1016/j.cpc.2011.06.002

References:
[1] J. Ellson, E. Gansner, E. Koutsofios, S. North, G. Woodhull, Graphviz and dynagraph – static and dynamic graph drawing tools, in: M. Junger, P. Mutzel (Eds.), Graph Drawing Software, Springer-Verlag, 2003, pp. 127–148.

1. Introduction

Random walks are useful in understanding fundamental classical processes such as diffusion and Brownian motion [1]. They have become a standard technique for modelling diverse processes in physics, chemistry, biology as well as in social science. They have also been applied to many different mathematical problems such as optimisation [2], solving differential equations [3], and investigating graph and network structure [4].

Quantum walks are a quantum mechanical extension of classical random walks [5]. However, they display remarkably different and non-intuitive dynamics, which have lead to the development of fast algorithms for various problems in the context of quantum computation, including search [6–8], element distinctness [9], testing group commutativity [10] and various graph-theoretic problems [11–14]. Quantum walks can also be regarded as a universal computational primitive, meaning that any computation can be formulated as a quantum walk on some graph [15,16].

In addition to their usefulness in an algorithmic context, quantum walks provide a flexible physical model of coherent or partially decoherent quantum systems [17] and tools developed in studies of quantum walks are beginning to be applied to study various problems in physics [18–20].

Early studies in the field focused on quantum walks on the line and were greatly aided by visualisations of how different initial states and walk parameters affect the probability distributions obtained [5,21,22]. More recently, studies have included quantum walks on higher-dimensional graphs [23]. As the structures considered have become more complex, the probability distributions have become increasingly difficult to visualise and interpret [24–32].

Unlike classical random walks, quantum walks on graphs display remarkably non-intuitive dynamics. A tool for interactive visualisation of the time-evolution of these quantum walks will provide insight into the relationship between the dynamics of a quantum walk and the underlying structure of the graph. This insight will be valuable in the development of quantum algorithms and in studying the various problems in physics that quantum walks are used to model. These visualisations will also allow results to be easily presented to other researchers, enabling rapid communication of ideas between research teams and also to broader audiences.

This paper is structured as follows. Section 2 begins with an introduction to the theoretical formulation of quantum walks. In Section 3, we provide an overview of the *qwViz* software and its installation. In Section 4, we give a more detailed description of the usage of *qwViz* including command line options and user interface. We also give some simple examples of results obtained using various inputs. Section 5 contains some examples of how the software can be used to intuitively visualise quantum walk data and investigate quantum walk dynamics. Finally, Section 6 contains our conclusions. Appendices A, B and C contain descriptions of the input and output file formats.

2. Discrete-time quantum walks on graphs

In this section we describe quantum walks taking place on graphs with an evolution in discrete time steps. We first define a graph in this context.

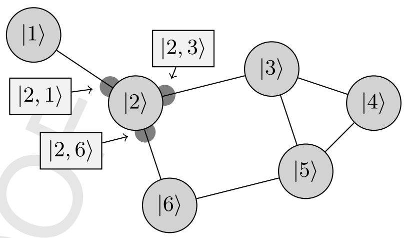

Fig. 1. Diagram of the vertex and subnode states of a 6-vertex undirected graph. Vertex states $|i\rangle$ for $i = 1,\ldots,6$ (circles) are shown for all vertices. The product states $|2,1\rangle$, $|2,3\rangle$ and $|2,6\rangle$ representing the subnodes of vertex 2 are also shown. The coin states of vertex 2 are labelled according to the label of the adjacent vertex.

Definition 1 *(Graph).* Let $G(V,E)$ be an undirected graph with vertex set $V = \{v_1, v_2, v_3, \ldots\}$ and edge set $E = \{(v_i, v_j), (v_k, v_l), \ldots\}$ consisting of unordered pairs of connected vertices. If there are d edges incident on a vertex v_i, we say that v_i has degree d. A graph G with N vertices is conveniently described by its adjacency matrix A, which is an $N \times N$ matrix satisfying,

$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The adjacency matrix of a graph is unique up to permutation and thus contains all information about the graph [33].

The vertices of a graph represent the quantum states that can be occupied throughout the evolution of the quantum walk. The edges connecting these vertices specify the permitted transitions between these quantum states at each time step. While the classical random walk on a graph is localised to a particular vertex at each time step, the quantum walk can be in a superposition of multiple vertex states, allowing the quantum walker to traverse multiple paths on the graph simultaneously throughout the evolution of the walk [23]. The probability distribution $P(t)$ over a vertex set V at a time t during the evolution of the quantum walk describes the probability of finding the quantum walker at each vertex $v_i \in V$ after t steps of the walk.

In order to describe the mathematical formalism of discrete-time quantum walks on graphs, we first define how the vertices and edges of a graph are related to the Hilbert space in which the quantum walk takes place. The vertices represent vertex states $\{|v_i\rangle\colon\ v_i \in V\}$, which form an orthonormal basis for the position Hilbert space $\mathcal{H}_\mathcal{P}$. Associated with each vertex v_i is also a set of "coin" states $\{|c_j\rangle\colon\ j = 1,\ldots,d_i\}$, where d_i is the degree of the vertex v_i. These coin states represent the outgoing edges of each vertex and span an auxiliary "coin" Hilbert space, $\mathcal{H}_\mathcal{C}$. The discrete-time quantum walk on a graph takes place on the *subnodes* of the graph, which are represented by product states of the form $|v\rangle \otimes |c\rangle = |v,c\rangle \in \mathcal{H}_\mathcal{P} \otimes \mathcal{H}_\mathcal{C}$ [6,23]. Fig. 1 shows a graph, with vertex and subnode states indicated.

Definition 2 *(Discrete-time quantum walk).* One step of the discrete-time quantum walk on a graph is the application of the unitary time-evolution operator $U = S \cdot (\mathbb{1} \otimes C)$, where S is the shift operator and C is the coin operator. S acts on the extended position space $\mathcal{H}_\mathcal{P} \otimes \mathcal{H}_\mathcal{C}$ as,

$$S|v_i, c_j\rangle = |v_j, c_i\rangle, \tag{2}$$

where $|v_i, c_j\rangle$ is the subnode state corresponding to the edge (v_i, v_j) at the vertex v_i. The coin operator C at a vertex v_i of degree d_i can be represented by a $d_i \times d_i$ matrix, which mixes the probability amplitudes of the subnode states of v_i [31,34].

A common choice of coin operator for quantum walks on graphs is the Grover coin G, which has elements [6,31,34]

$$G_{ij} = -\delta_{ij} + 2/d. \qquad (3)$$

3. Overview of the software

When executed, *qwViz* can perform one of two distinct tasks. Firstly, it can compute and plot time-dependent probability distributions for discrete-time quantum walks on graphs. In this case, the input data is simply the adjacency matrix of the graph, specified as an ADJ file (Appendix A). Parameters controlling the quantum walk can be manipulated using command line options. In this way, *qwViz* provides the ability to quickly examine a quantum walk or quantum-walk-based spatial search procedure on a particular structure. There are, however, limited built-in options available for the customisation of the walks, making direct simulation by *qwViz* insufficient for researchers who wish to examine the effects of various initial states, coin operators and decoherence on the dynamics of quantum walk.

Secondly, to allow users to have complete control over the mathematical operations involved in the quantum walk, *qwViz* can be used to plot externally generated quantum walk data. An XML-based file format QWML (Quantum Walk Markup Language) is introduced here to facilitate the input of data from quantum walk simulations into *qwViz* (Appendix B). It should be noted that QWML files can be used to visualise arbitrary probability distributions, including those arising from continuous-time quantum walks and even classical random walks.

To facilitate simple export of quantum walk simulation data from user generated software, functions written in C and Fortran for writing QWML files are provided with the *qwViz* package.

For both the ADJ and QWML inputs, visualisation commences once the data structures required to plot the probability distribution are computed (or read from a file) and all vertices of the graph are assigned coordinates in two dimensions. Straight edges connecting these vertices are drawn and the probability distribution is plotted at each time step using cylinders at each vertex, whose height represents the probability of finding the quantum walker at that particular vertex. The cylinders are also coloured according to the value of probability that they represent.

3.1. Installation and dependencies

The *qwViz* package is simple to install on Mac OS X and Linux environments. The package uses OpenGL and GLUT libraries for handling graphics and Graphviz libraries for generating two-dimensional graph layouts from graph structural information provided at run-time. *Graphviz* [35] source code is freely available under the Eclipse Public License version 1.0 (www.graphviz.org). Standard installations of *Graphviz* from source code or using MacPorts place the required libraries in directories which will be automatically linked by the standard *qwViz* installation. For users who do not install *Graphviz* in the standard directories, more information is provided with the *qwViz* source code.

The two input file formats ADJ and QWML are described in Appendices A and B, respectively. Once the program is installed and the directory containing the `qwViz` binary has been added to the path, one just needs to type

```
qwViz infile.adj
```

or

```
qwViz infile.qwml
```

to execute the program. The `.adj` or `.qwml` filename extension for the input data file is used to determine whether data should be generated based on command line controls (ADJ) or whether externally generated data are being plotted.

3.2. Performance

For all cases tested (up to 1331 vertices), the software ran at the target frame rate of 60 fps, which allowed realtime manipulation of the visualisation. While TIFF or TGA images are being continuously exported, the software may run at a lower frame rate depending on the size of the image window and relative speed of the hard disk.

4. Usage

We now describe the usage of *qwViz* with ADJ (Section 4.1) or QWML (Section 4.2) input files. Command line options and interactive user interface which are common to both input formats are described in Section 4.3. The input files for all examples presented are provided with the source code.

4.1. Adjacency matrix inputs: generating quantum walk data

The time-dependent probability distribution for a discrete-time quantum walk on a graph can be directly computed by providing the adjacency matrix of the graph as an ADJ input file. The ADJ file format is described in Appendix A. *Graphviz* library functions are called to generate a two-dimensional graph layout based on the connectivity information provided by the adjacency matrix.

4.1.1. Options

We now explain in detail the command line options that can be used to control the simulation of discrete-time quantum walks with ADJ file inputs. It should be noted that `-start` and `-search` options are not compatible options since their initial states differ. The numbering of the vertices is obtained from the row numbers of the adjacency matrix from 1 to N.

1. `-start` v_i, where $1 \leqslant v_i \leqslant N$ is an integer specifying the vertex at which the quantum walker is initially localised. The quantum walk procedure is explained in Section 4.1.2. The default starting vertex is vertex 1.
2. `-search` v_i, where $1 \leqslant v_i \leqslant N$ is an integer specifying the marked vertex in the quantum-walk-based search procedure described in Section 4.1.3.
3. `-steps` t, where $t > 0$ is an integer specifying the number of steps in the quantum walk. The default number of steps in a simulation is 200.
4. `-o` *outfile*, where *outfile* is a filename with extension `.qwml` or `.prob`. This option causes the time-dependent probability distribution computed to be written to an external file. The file format is determined from the filename extension. The QWML and PROB file formats are described in Appendices A and C, respectively.

If no command line options are specified then the default procedure is to simulate a 200-step quantum walk from vertex 1, which is equivalent to typing,

```
qwViz -start 1 -steps 200 infile.adj
```

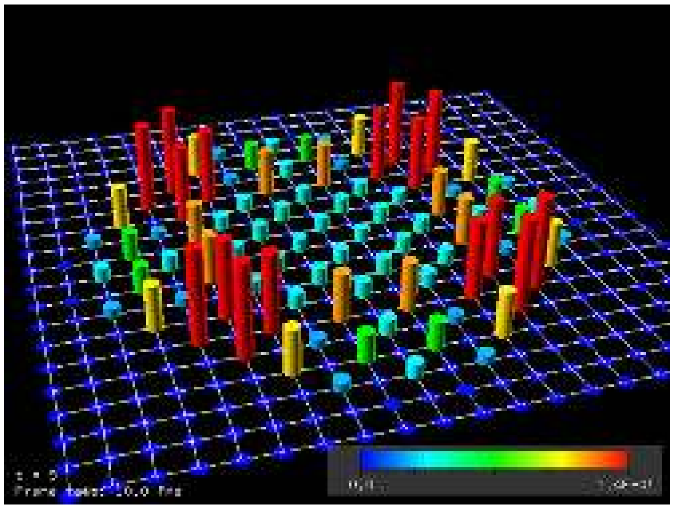

Fig. 2. Single frame of the visualisation generated by simulating a quantum walk with Grover coin from the central vertex (vertex 210) of a 20×20 two-dimensional lattice, based on the adjacency matrix given in `grid20x20.adj`. The coordinates of the vertices are automatically generated using the *Graphviz*' *neato* algorithm.

4.1.2. Simulating quantum walks

The initial state $|\psi_0\rangle$ for the simulations of quantum walks on graphs is an equal superposition of the coin states of the starting vertex

$$|\psi_0\rangle = \sum_{c_i=1}^{d_i} |v_i, c_i\rangle. \tag{4}$$

The quantum walk proceeds from this state by the repeated application of the unitary time-evolution operator U, as described in Section 2. The coin operator at each vertex is the Grover coin (Eq. (3)).

The ADJ file `grid20x20.adj` contains the adjacency matrix of a 20×20 two-dimensional regular lattice. The command

```
qwViz -start 210 grid20x20.adj
```

computes and plots the time-dependent probability distribution arising from a quantum walk starting at vertex 210 (a central vertex). A plot of the probability distribution after 9 steps of the walk is shown in Fig. 2.

4.1.3. Simulating quantum-walk-based search

The initial state $|\psi_0\rangle$ for the simulations of quantum-walk-based search on graphs is an equal superposition of all vertex states $|v_i\rangle \in \mathcal{H}_\mathcal{P}$. The probability amplitude at each vertex is then divided equally between all subnodes. Formally, the initial state is given by

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} \sum_{j=1}^{d_i} \frac{1}{\sqrt{d_i}} |v_i, c_j\rangle. \tag{5}$$

The target of the search procedure is called the *marked vertex*, where the marking is intended to represent a "quantum oracle" and is implemented as a perturbation to the coin operator at the marked vertices. The quantum search procedure proceeds via the repeated application of the perturbed time-evolution operator, $U' = S \cdot (\mathbb{1} \otimes C')$ where the coin operator at vertex v_i is given by

$$(C'_i)_{mn} = \begin{cases} -\delta_{mn} + 2/d_i, & v_i \notin M, \\ -\delta_{mn}, & v_i \in M, \end{cases}$$
$$\text{for } m, n = 1, \ldots, d_i. \tag{6}$$

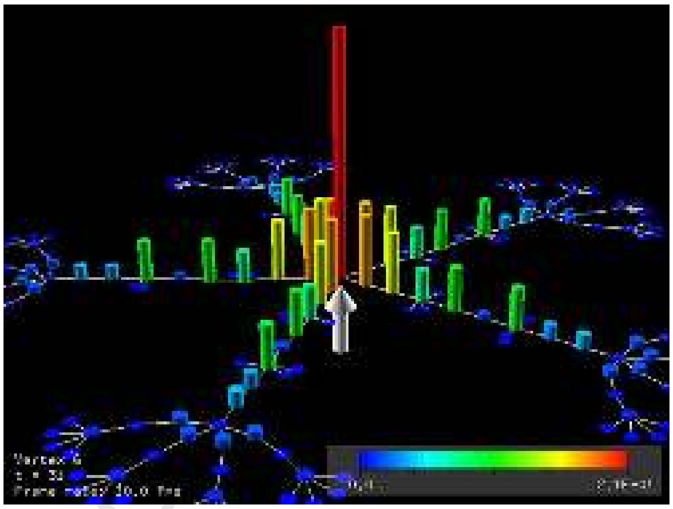

Fig. 3. Single frame of the visualisation generated by simulating the quantum-walk-based search procedure described in Section 4.1.3 on the regular hyperbranched fractal given in `RHFf5g3.adj`. The coordinates of the vertices are automatically generated using *Graphviz*' *neato* algorithm. The central vertex (vertex 6) is "marked" in the search procedure and is indicated with the vertex identification arrow.

Depending on the structure of the graph and the position of the marked vertex within the structure, this quantum search procedure can result in significant amplification of the probability of finding the quantum walker at the marked vertex, and can in some cases lead directly to a quantum-walk-based algorithm for spatial search [6].

Quantum walks and quantum-walk-based search on regular hyperbranched fractals have been studied in [30,31]. The ADJ file `RHFf5g3.adj` contains the adjacency matrix of a third generation regular hyperbranched fractal, with a functionality of 5. The command

```
qwViz -search 6 RHFf5g3.adj
```

computes and plots the time-dependent probability distribution arising from the search procedure described above with vertex 6 (the central vertex) being marked. The probability of finding the particle at the central vertex reaches its first maximum after 31 steps of the walk (Fig. 3).

4.2. QWML inputs: plotting existing quantum walk data

In order to plot externally generated quantum walk data, it is necessary to export the adjacency matrix and the time dependent probability distribution as a QWML file. Appendix B contains a description of the QWML file format.

Upon executing `qwViz` with a QWML input file. The data structures required to plot the quantum walk data are populated according to the input file. If the `<graphlayout>` tag is present in the QWML input file and (x, y)-coordinates are successfully read for all vertices, then the vertices will be laid out according to the coordinates given in the file. Otherwise *Graphviz* library functions will be called to generate a two-dimensional layout of the graph and assign (x, y)-coordinates to the vertices.

The QWML file `dsg3.qwml` contains the adjacency matrix, probability distribution and graph layout information for a discrete-time quantum walk on a third generation dual Sierpiński gasket starting at a peripheral vertex, equivalent to data presented in [29]. Typing

```
qwViz dsg3.qwml
```

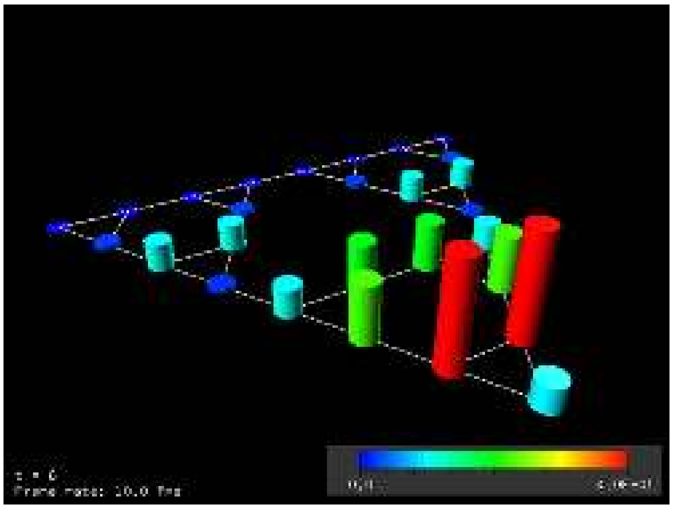

Fig. 4. Single frame of the visualisation generated from the `dsg3.qwml` file. In this case the coordinates of the vertices are specified in the QWML file.

produces a time-dependent visualisation of the quantum walk data contained in `dsg3.qwml`. A frame of this visualisation with after 6 steps of the walk is shown in Fig. 4.

4.3. *Command line options and interactive user interface*

The command line options listed below are available when using either ADJ or QWML input files.

1. `-h`, prints help text to standard error.
2. `-v`, prints messages about the program to standard error while the program is running (verbose mode).
3. `-f`, resizes the display window to take up the full screen.
4. `-s`, active stereo mode.
5. `-ss`, dual (side-by-side) stereo mode.
6. `-bg` $r\ g\ b$, where $r, g, b \geqslant 0$ are floating point numbers. Sets the background colour (default is black: `-bg 0 0 0`).
7. `-cs` i, where $1 \leqslant i \leqslant 25$ is an integer. Sets the colour scheme for the probability distribution (default is `-cs 1`).
8. `-neato`, uses *Graphviz*' *neato* algorithm to layout the vertices. This is the default layout algorithm and typically achieves the best results.
9. `-circo`, uses *Graphviz*' *circo* algorithm to layout the vertices in a circle.
10. `-fdp`, uses *Graphviz*' *fdp* (Fruchterman–Reingold) algorithm to layout the vertices.
11. `-tiff`, changes the image export format to TIFF (default is TGA).
12. `-i` *subframes*, where *subframes* > 1 is an integer. Linearly interpolates the data between time steps to plot *subframes* frames per data point. This results smoother growing and shrinking of the cylinders representing the probability distribution. By default this option is disabled so only the data for integer time steps is plotted.

The interactive user interface is shown in Table 1. The keys indicated in the Key/Mouse button column of Table 1 are used for real-time manipulation of the visualisation.

5. Examples of usage

We now demonstrate how *qwViz* can be used to visualise a well-known result obtained for quantum walks on graphs. In 2003, Childs et al. [11] constructed an artificial problem that can be solved exponentially faster by quantum walks than any known classical method. The problem takes place on the *glued-trees* graph, which is constructed by randomly connecting the leaves of two balanced binary trees (Fig. 5). The problem is to start at one root vertex (entrance) and to find the other (exit) using the minimum number of steps. This problem has also been studied numerically for discrete-time quantum walks [24].

Fig. 5 shows the probability distributions of (a) a random walk and (b) a quantum walk after 9 steps, starting from the entrance vertex. As shown in Fig. 5, the quantum walk has a far greater probability of being found at the exit vertex than the random walk. For the initial state studied here, it is possible to project

Table 1
Description of the user interface options.

Key/Mouse button	Description
Camera	
`< >`	Move camera forward or backward
`+ -`	Zoom in/out by changing aperture
`[ ]`	Roll camera (anticlockwise/clockwise)
Arrow keys or left mouse	Rotate camera
Shift + Arrow keys or left mouse	Translate camera
`a A z`	Automatic rotation (anticlockwise/clockwise/stop)
`x`	Return camera to home position
`f1 ... f6`	Special camera views
Data visualisation	
`p`	Toggle play/paused
`n b`	Next/back: increment or decrement time
`/`	Reset time to zero
`u d`	Up/down: adjust the scale for the plotted data
`s f`	Slow/fast: adjust the frame rate
`c v`	Adjust the radius of the vertices (increase/decrease)
Vertex identification	
`j`	Toggle vertex identification arrow on/off
`k l`	Increment the vertex identification arrow
Other	
Right mouse	Popup menus
`w`	Window dump to TGA file in working directory
`r`	Record: Toggle continuous window dump
`h`	Help: Display help text and interface information
`i`	Info: Display scale and time step information

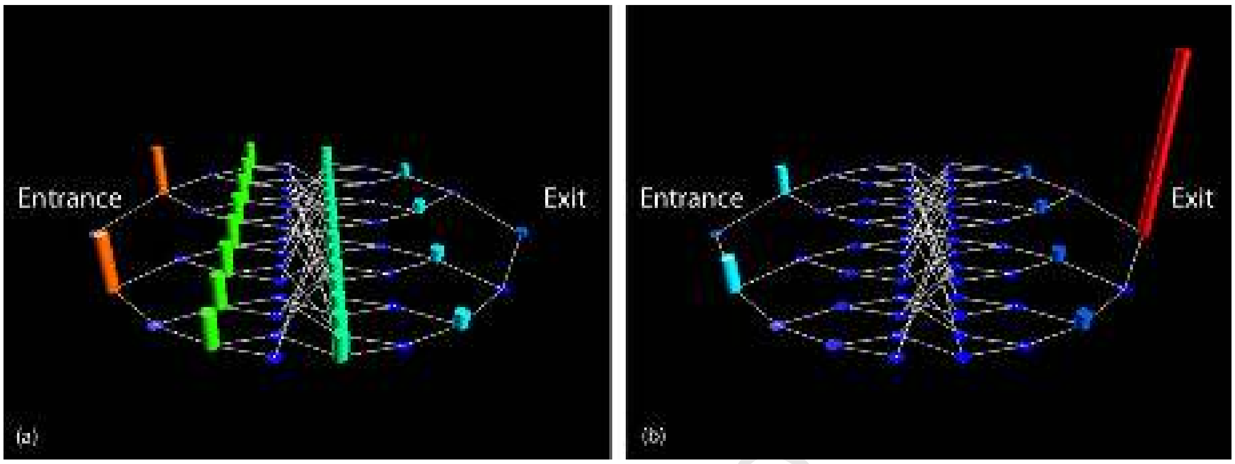

Fig. 5. A comparison of the probability distributions obtained after 9 steps of (a) classical random walk and (b) quantum walk starting from the entrance node.

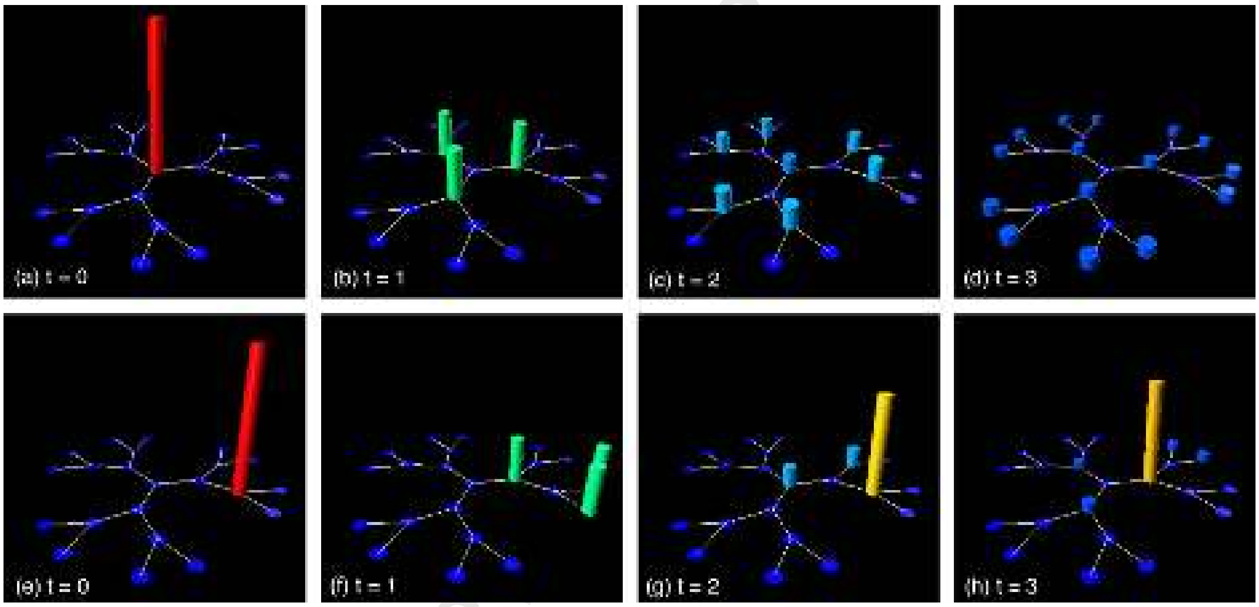

Fig. 6. A comparison of the probability distributions obtained from the first three steps of an unbiased quantum walk on a Cayley tree starting from (a)–(d) a central vertex or (e)–(h) a non-central vertex. The symmetric distribution obtained in the central case is relatively intuitive compared to the distribution obtained from the non-central case however *qwViz* allows both cases to be easily investigated.

the glued trees problem onto a line and solve analytically for the time-evolution of the walk. However, using *qwViz* to view the time-evolution of the probability distribution on the graph provides a more intuitive picture of what is actually happening and allows the user to view the probabilities of all states simultaneously. Moreover, using *qwViz*, quantum walks from any initial state can be studied in the same manner, not just from initial states with particular symmetry.

We now examine the case of quantum walks and quantum-walk-based search on Cayley trees, which have also been studied previously [26,31,32,36]. Quantum walks with both central and non-central starting positions have been studied numerically as a model of coherent exciton transport [26], though visualisation of results has been difficult for non-central excitations since the rotational symmetry often used to reduce the complexity of the problem is broken. Analytical results have also been obtained for search on Cayley trees when the central vertex is marked, but not for any other positions of the marked vertex [31]. Using *qwViz* to visualise quantum walks and search on Cayley trees allows probability distributions from non-central starting positions to be visualised in exactly the same manner as those for central starting positions. Fig. 6 shows the first few steps of quantum walk with initial central and non-central excitations. Visualising quantum walk data for less symmetric initial states will lead to a greater understanding of the dynamics of quantum walks, which may lead to the development of quantum-walk-based algorithms on more complex structures.

Finally, we give an example of how *qwViz* can be used to investigate quantum walk dynamics on large non-vertex-transitive structures, such as a fifth generation Sierpiński gasket, which has 366 vertices. Shown in Fig. 7 are two frames from the visualisation of the quantum-walk-based search for a peripheral vertex on the Sierpiński gasket. Although Fig. 7 gives some indication of the complexity of the dynamics of quantum walks on large non-vertex-transitive structures, viewing the full visualisations produced by *qwViz*, it is possible to see the probability distribution spreading over the surface of the graph and directly observe the various frequencies involved. The time-dependent visualisations also make it possible to see how certain structural features of the graph influence the flow of probability distribution around the structure. For example, we can see that the vertices which connect the larger triangles of the Sierpiński gasket have, on average, higher prob-

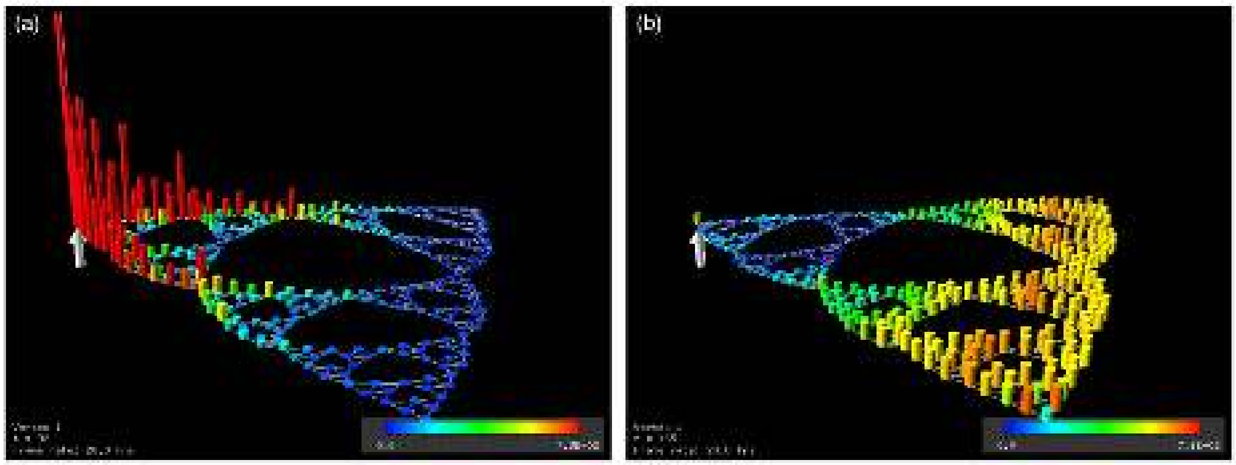

Fig. 7. Visualising the quantum-walk-based search for marked vertex 1 (indicated with the arrow at the far left of the graph); (a) shows the probability distribution after 92 steps of the walk when the probability of finding the quantum walker at the marked vertex is a maximum; (b) shows the probability distribution after ~~167~~ steps of the walk. Observing the probability distribution flow over the surface of this graph throughout this simulation provides an intuitive picture of large scale quantum walk dynamics.

abilities than other vertices because the probability distribution must flow through these vertices to access the different areas of the graph.

6. Conclusions

We have developed *qwViz* as a tool for the interactive visualisation of the time evolution of probability distributions arising from quantum walks on graphs. The software can function by direct simulation of discrete-time quantum walks on graphs or by plotting externally data generated data arising from discrete- or continuous-time quantum or random walks. This allows the user to either perform simple investigations using the built-in simulation capability or to have complete control over the mathematical operations involved in the quantum walk, making the software suitable for research in the fields of quantum-walk-based algorithm development and using quantum walks to model physical systems. Two-dimensional graph layouts can either be automatically generated or alternatively, the coordinates of each vertex can be specified by the user.

We have demonstrated that *qwViz* allows intuitive realtime visualisation of results that were previously difficult to interpret and provides insight into quantum walks on large non-vertex-transitive structures where intuitive visualisation of simulation data has been previously impossible. Researchers interested in using quantum walks as a physical model to study coherent quantum systems such as exciton transport on structures like the Fenna–Matthews–Olson protein complex [37], will find such visualisations useful in analysing results of numerical simulations.

Acknowledgements

This work was supported by iVEC through the use of advanced computing resources located at The University of Western Australia.

Appendix A. ADJ file format

The ADJ file format is used to store the adjacency matrices of graphs in a human-readable form. ADJ text files may contain a header with the name of the graph and any other text. The adjacency matrix is read from the file when either “0” or “1” is encountered as the first character in the line, for this reason the header may not begin any line with the characters “0” or “1”. Note that *qwViz* does not allow quantum walk data to be calculated for graphs with multiple or directed edges or self-loops. For a graph with N vertices, the entries in the first row take up the first N characters of each line, whitespace may follow each row of the adjacency matrix. The number of digits in the first row is used to allocate memory to store the entire adjacency matrix. Successive rows of the matrix must be separated by new lines. An ADJ file is shown in Fig. 8.

```
Complete graph (N=4)

0111
1011
1101
1110
```

Fig. 8. The contents of the file `complete4.adj`, containing the adjacency matrix of the complete graph with four vertices.

ADJ files can be automatically produced from graph6 or sparse6 files using the *showg* utility, which comes as part of the *nauty* package [38]. If *showg* is installed, typing

```
showg -a graphfile.g6 > graphfile.adj
```

will convert graphfile.g6 to a valid ADJ file.

Appendix B. QWML file format

The QWML file format conforms to the W3C Extensible Markup Language (XML) 1.0 specification [39]. An XML Schema (`qwml.xsd`) is provided with the source code. The root tag for QWML filed is `<qwml>` and at the first level there are two required tags, `<adjacency>` and `<probdist>` and the optional tag `<graph-layout>`. At this level, other tags are allowed but will be ignored by *qwViz*. Whitespace is permitted inside tags and data fields but is not advised. Blank lines are also permitted but not advised. Several example QWML files are included with the source code.

The `<adjacency>` tag contains only `<row>` tags, which in turn contain the elements `<col>`, providing the adjacency matrix data. An example of an `<adjacency>` structure is shown in Fig. 9(a).

The `<probdist>` tag contains the probability distribution, ordered by `<vertex>` tags. In turn, these `<vertex>` tags contain the elements `<prob>`, which provide the time-ordered probabilities of finding the quantum walker at a vertex. These are read by *qwViz* as double precision floating point numbers. The number of

(a)
```
<adjacency>
 <row>
  <col>0</col>
  <col>1</col>
 </row>
 <row>
  <col>1</col>
  <col>0</col>
 </row>
</adjacency>
```

(b)
```
<probdist>
 <vertex>
  <prob>0.33333333</prob>
  <prob>0.67847367</prob>
  <prob>0.33333333</prob>
 </vertex>
 <vertex>
  <prob>0.66666666</prob>
  <prob>0.32152633</prob>
  <prob>0.66666666</prob>
 </vertex>
</probdist>
```

Fig. 9. Example of (a) the adjacency matrix of a two-vertex connected graph as written under the `<adjacency>` structure of a QWML file and (b) the probability distribution for the first two steps of a quantum walk on this graph as written under the `<probdist>` structure.

```
<graphlayout>
 <vertex>
  <xcoord>0</xcoord>
  <ycoord>1</ycoord>
 </vertex>
 <vertex>
  <xcoord>1</xcoord>
  <ycoord>0</ycoord>
 </vertex>
</graphlayout>
```

Fig. 10. Example of a graph layout for a two-vertex connected graph as written under the `<graphlayout>` tag of a QWML file.

`<prob>` tags inside the first `<vertex>` tag determines the number of steps in the quantum walk. All subsequent `<vertex>` tags must contain the same number of `<prob>` tags. The order of the vertices in `<probdist>` must be the same as that given by the adjacency matrix. An example is shown in Fig. 9(b).

Like `<probdist>`, `<graphlayout>` is a simple type containing `<vertex>` tags, which themselves contain the sequence of elements `<xcoord>` and `<ycoord>` to provide the coordinates of vertices in two dimensions. The data for the `<xcoord>` and `<ycoord>` elements must be single-precision floating point numbers. The coordinates will be read by *qwViz* and scaled to fit in the viewing area. An example is shown in Fig. 10. As for the `<probdist>` tag, the vertices must be in the same order as the adjacency matrix.

Appendix C. PROB file format

A PROB output file is designated by the `.prob` filename extension and contains a table of double precision floating point numbers representing the probabilities of finding the quantum walker at each vertex. The columns follow the order of the vertices in the adjacency matrix and successive rows of the table represent successive steps of the walk. The PROB file format provides the ability to export quantum walk simulation data from *qwViz* in a table-based form which can be readily imported by other software packages.

References

[1] D. ben Avraham, S. Havlin, Diffusion and Reactions in Fractals and Disordered Systems, Cambridge University Press, Cambridge, UK, 2000.
[2] B.A. Berg, Nature 361 (1993) 708.
[3] S. Hoshino, K. Ichida, Numer. Math. 18 (1971) 61.
[4] J.D. Noh, H. Rieger, Phys. Rev. Lett. 92 (2004) 118701.
[5] J. Kempe, Contemp. Phys. 44 (2003) 307.
[6] N. Shenvi, J. Kempe, K.B. Whaley, Phys. Rev. A 67 (2003) 052307.
[7] A.M. Childs, J. Goldstone, Phys. Rev. A 70 (2004) 022314.
[8] A. Tulsi, Phys. Rev. A 78 (2008) 012310.
[9] A. Ambainis, Quantum walk algorithm for element distinctness, in: FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, USA, 2004, pp. 22–31.
[10] F. Magniez, A. Nayak, Algorithmica 48 (2007) 221, doi:10.1007/s00453-007-0057-8.
[11] A.M. Childs, et al., Exponential algorithmic speedup by a quantum walk, in: STOC '03: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, 2003, pp. 59–68.
[12] B.L. Douglas, J.B. Wang, J. Phys. A: Math. Theor. 41 (2008) 075303.
[13] J.K. Gamble, M. Friesen, D. Zhou, R. Joynt, S.N. Coppersmith, Phys. Rev. A 81 (2010) 052313.
[14] M. Hillery, D. Reitzner, V. Bužek, Phys. Rev. A 81 (2010) 062324.
[15] A.M. Childs, Phys. Rev. Lett. 102 (2009) 180501.
[16] N.B. Lovett, S. Cooper, M. Everitt, M. Trevers, V. Kendon, Phys. Rev. A 81 (2010) 042330.
[17] V. Kendon, Math. Structures Comput. Sci. 17 (2007) 1169.
[18] M. Katori, S. Fujino, N. Konno, Phys. Rev. A 72 (2005) 012316.
[19] T. Oka, N. Konno, R. Arita, H. Aoki, Phys. Rev. Lett. 94 (2005) 100602.
[20] P. Kurzynski, Phys. Lett. A 372 (2008) 6125.
[21] T.A. Brun, H.A. Carteret, A. Ambainis, Phys. Rev. A 67 (2003) 052317.
[22] B. Tregenna, W. Flanagan, R. Maile, V. Kendon, New J. Phys. 5 (2003) 83.
[23] D. Aharonov, A. Ambainis, J. Kempe, U. Vazirani, Quantum walks on graphs, in: STOC '01: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, 2001, pp. 50–59.
[24] I. Carneiro, et al., New J. Phys. 7 (2005) 156.
[25] O. Mülken, A. Blumen, Phys. Rev. E 71 (2005) 016101.
[26] O. Mülken, V. Bierbaum, A. Blumen, J. Chem. Phys. 124 (2006) 124905.
[27] A. Blumen, V. Bierbaum, O. Mülken, Phys. A: Statist. Mech. Appl. 371 (2006) 10.
[28] O. Mülken, V. Pernice, A. Blumen, Phys. Rev. E 76 (2007) 051125.
[29] E. Agliari, A. Blumen, O. Mülken, J. Phys. A: Math. Theor. 41 (2008) 445301.
[30] A. Volta, J. Phys. A: Math. Theor. 42 (2009) 225003.
[31] S.D. Berry, J.B. Wang, Phys. Rev. A 82 (2010) 042333.
[32] E. Agliari, A. Blumen, O. Mülken, Phys. Rev. A 82 (2010) 012305.
[33] C. Godsil, G. Royle, Algebraic Graph Theory, Springer, New York, NY, USA, 2001.
[34] A. Ambainis, J. Kempe, A. Rivosh, Coins make quantum walks faster, in: SODA '05: Proceedings of the Sixteenth Annual ACM–SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005, pp. 1099–1108.
[35] J. Ellson, E. Gansner, E. Koutsofios, S. North, G. Woodhull, Graphviz and dynagraph – static and dynamic graph drawing tools, in: M. Junger, P. Mutzel (Eds.), Graph Drawing Software, Springer-Verlag, 2003, pp. 127–148.
[36] K. Chisaki, M. Hamada, N. Konno, E. Segawa, Interdisciplinary Inform. Sci. 15 (2009) 423.
[37] M. Mohseni, P. Rebentrost, S. Lloyd, A.A. Guzik, J. Chem. Phys. 129 (2008) 174106.
[38] B.D. McKay, nauty User's Guide (Version 2.4), Australian National University, Canberra, Australia, 2009.
[39] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible markup language (XML) 1.0, fifth edition, Technical report, W3C, 2008.