Solid Earth, 5, 741–755, 2014
www.solid-earth.net/5/741/2014/
doi:10.5194/se-5-741-2014

Plate tectonic raster reconstruction in GPlates

J. Cannon[1], **E. Lau**[1,*], **and R. D. Müller**[1]

[1]EarthByte Group, School of Geosciences, University of Sydney, Australia
[*]now at: Google, Sydney, Australia

Correspondence to: J. Cannon (john.cannon@sydney.edu.au)

Received: 18 February 2014 – Published in Solid Earth Discuss.: 12 March 2014
Revised: 13 June 2014 – Accepted: 17 June 2014 – Published: 1 August 2014

Abstract. We describe a novel method implemented in the GPlates plate tectonic reconstruction software to interactively reconstruct arbitrarily high-resolution raster data to past geological times using a rotation model. The approach is based on the projection of geo-referenced raster data into a cube map followed by a reverse projection onto rotated tectonic plates on the surface of the globe. This decouples the rendering of a geo-referenced raster from its reconstruction, providing a number of benefits including a simple implementation and the ability to combine rasters with different geo-referencing or inbuilt raster projections. The cube map projection is accelerated by graphics hardware in a wide variety of computer systems manufactured over the last decade. Furthermore, by integrating a multi-resolution tile partitioning into the cube map we can provide on-demand tile streaming, level-of-detail rendering and hierarchical visibility culling, enabling researchers to visually explore essentially unlimited resolution geophysical raster data attached to tectonic plates and reconstructed through geological time. This capability forms the basis for interactively building and improving plate reconstructions in an iterative fashion, particularly for tectonically complex regions.

1 Introduction

Geospatial information system (GIS) software enables geoscientists to visually explore vector and raster geospatial data. In the plate tectonics domain there is also the need to explore data in their spatial arrangement through geological time. GPlates (Boyden et al., 2011) is open-source software that combines familiar GIS functionality with the time dimension enabling visual and analytical exploration of spatio-temporal relationships within geospatial data. Typical geospatial data consists of present-day observations on tectonic plates. Due to the movement of plates throughout geological history this data must be *reconstructed* from its present-day configuration to its spatial arrangement at past geological times before spatio-temporal exploration can occur. This is achieved by attaching present-day vector geometry and raster data to tectonic plates on the surface of the globe and rotating them to past geological configurations using a plate tectonic rotation model. The rotation model provides rotations for each tectonic plate over a period of geological history. Each rotation, consisting of an axis passing through the globe centre and an angle, rigidly rotates a tectonic plate across the spherical surface of the globe. The need for interactive raster reconstruction includes the ability to quickly reconstruct high-resolution geophysical raster data within alternative plate tectonic rotation models in order to build, alter and optimize, or simply visualize and interrogate those models (Williams et al., 2012). Part of this involves visualizing how well ancient geological features align at reconstruction times when they were in close proximity to each other. Here we focus on the implementation of raster reconstruction within GPlates. Reconstructing raster data involves correctly positioning a raster on the globe (geo-referencing), partitioning it into tectonic regions and independently rotating each tectonic region according to its motion through geological time. Interactive exploration of raster data, at the fixed resolution of the computer monitor screen, requires the user to pan and zoom the view in order to expose desired raster regions and details. Since the imported raster data can have arbitrarily high resolution we must employ visibility culling and level-of-detail (LOD) techniques. These enable the raster to be efficiently rendered at the highest detail level permitted by the monitor resolution and the user's zoom level. Firstly, LOD reduces the workload by down-sampling the original raster image into successively less-detailed images and only loading and rendering raster data at a level of detail sufficient to satisfy the pixel resolution of the computer monitor. And secondly, visibility culling reduces the workload by spatially partitioning each LOD image into tiles. Only those tiles that are visible for a given view position and direction are then loaded and rendered. However, we also need to integrate LOD and visibility culling with interactive reconstruction of raster data (attached to independently rotating tectonic blocks). Our approach is based on a cube map projection of the globe whereby raster pixels are radially projected from the surface of the globe onto a cube surrounding the globe and where each face of the cube is partitioned into a LOD hierarchy of image tiles. With the cube map approach the rendering of a reconstructed raster follows a decoupled two-stage process. Both stages take advantage of the ability of graphics hardware to rapidly project and render a 3-D scene into a frame buffer (or target texture). The first stage renders the geo-referenced raster into one or more tile textures of the cube map. The second stage involves unprojecting those cube map tile textures back onto the globe during the rendering of rotated tectonic surface regions. This separation of rendering stages simplifies the implementation of raster reconstruction on the 3-D globe while also enabling enhancements such as raster reconstruction in different 2-D map projections and combining multiple rasters. Virtually all graphics hardware manufactured in the last decade has dedicated hardware to accelerate transformations using 4×4 matrices. The cube map approach takes advantage of this since each individual cube map tile has its own cube map projection transform represented as an off-axis perspective 4×4 matrix. This enables GPlates to reconstruct and display arbitrary resolution raster data at interactive frame rates on a wide variety of computer systems.

2 Multi-resolution raster visualization

GPlates initially uses a multi-resolution raster rendering approach when data are loaded into GPlates, i.e. when the user has chosen not or not yet to reconstruct a raster. Additionally, when the user does choose to reconstruct a raster, this approach is used as the first stage of the raster reconstruction process (to render that raster into a cube map).

2.1 Source raster data

In order to display raster data in GPlates a raster file needs to be imported. The import procedure generates a GPlates Markup Language (GPML) file with a single raster feature containing a link to the original raster image file and raster meta-information including geo-referencing, coordinate reference system and general raster metadata (Qin et al., 2012). GPlates uses the Geospatial Data Abstraction Library (GDAL), an open-source library for raster geospatial data formats (Warmerdam, 2008), for importing raster data. Source raster data from a variety of colour image formats containing Red Green Blue Alpha (RGBA) colour data (including JPEG and PNG) can be imported into GPlates. Also, a variety of numerical image formats containing floating-point or integer data (including NetCDF and GeoTIFF) can be imported. Both types of raster data can be visualized in RGBA format – for numerical raster data (which has no colour information) the user also selects a colour palette to translate from numerical pixels to RGBA pixels.

2.2 Rationale for multi-resolution raster data

GPlates takes advantage of the power of graphics hardware to achieve raster rendering at interactive frame rates. However, the raster data still needs to be retrieved from storage before it can be rendered. Since this is usually a hard disk drive (HDD) with mechanical disk platters this is by far the slowest component in raster visualization. More recently solid state drives (SSD) are becoming less expensive and their random access reads have much lower latencies than in HDDs. However, HDD and SSD bandwidths are still comparable so loading the entire raster into system memory is not desirable since this can consume an excessive amount of time and computer memory. Especially since GPlates supports arbitrarily high raster image resolutions (such as one minute global grids and higher) which can exceed the maximum of 3–4 GB of virtual memory addressable by a 32 bit application (even if the system has much more physical memory).

Therefore, it is important to only interactively load raster data that is both (i) at a resolution closely matching that of the desktop monitor (or visual display screen), and (ii) that is visible in the view window. This level of detail and visibility culling is achieved with a multi-resolution tiled raster image pyramid that significantly reduces the amount of raster data streamed from disk and reduces the rendering workload for the graphics hardware.

2.3 Multi-resolution tiled raster image pyramid

The raster image pyramid is generated by successively filtering the full resolution raster image into progressively smaller versions of itself with each pyramid level halving the dimension of the previous level (for example 4000×2000, 2000×1000, 1000×500, etc.). Since this filtering is generally expensive it is only done once and written to a cache file (when the raster is first rendered) such that subsequent sessions use the cache file directly. It should be noted that this cache file resides on disk and not in memory.

During raster rendering GPlates selects the lowest resolution image that achieves a sharp and crisp visual appearance of the raster (of course, when even the highest-detail resolution of the raster is insufficient, the raster cannot appear sharp and crisp). For example, if the entire raster is visible in the view window (when the view is fully zoomed out) and the

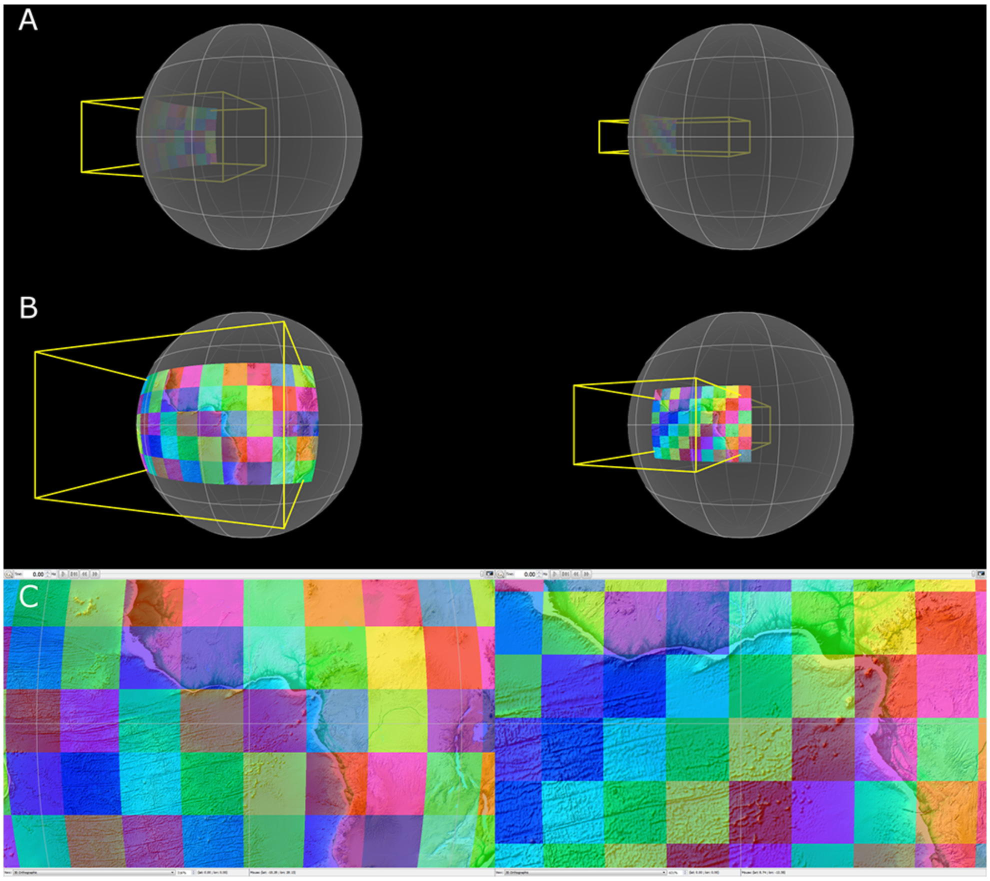

Figure 1. Visible geo-referenced tiles of the ETOPO1 (1 Arc-Minute Global Relief Model) raster (Amante and Eakins, 2009) highlighted with solid colours. The left and right sides show two different zoom levels. The yellow wireframe is the orthographic-(rectangular prism) view frustum used by the "3-D Orthographic" view in GPlates. Note that **(A–B)** on left side show the same zoomed-out view (yellow frustum) but from different angles – similarly for **(A–B)** on the right with the zoomed-in view. **(A)** more clearly shows that the yellow frustum contains only the front surface of the globe, rejecting the rear surface; **(B)** more clearly shows which geo-referenced tiles overlap the view frustum (yellow frustum) – note that tiles outside the frustum are culled; **(C)** The view as seen by the user of GPlates (the region inside the yellow frustum).

full-resolution raster (level 0) has dimensions $21\,600 \times 10\,800$ (a 1 min global grid) and the display screen has dimensions 1650×1050, then the down-sampled image at level 3 in the pyramid (of dimensions 2700×1350) is sufficient to display the raster without visual loss of detail.

In the above example the view was fully zoomed out to display the entire globe (or raster) requiring only a low-resolution raster pyramid level to be loaded and rendered. The opposite scenario occurs when the view is fully zoomed in to observe, in detail, a specific localized geographic region. Here a much higher raster level of detail is required but a correspondingly smaller area of the globe (or raster) is visible in the view window – essentially the majority of the raster is not actually visible. To enable visibility culling, each image in the pyramid is partitioned into small fixed-size tiles and only the visible tiles are loaded and rendered. Note that all tiles contain the same number of pixels regardless of their pyramid level and hence place the same load on the system (graphics hardware and streaming).

Interactive visualization is then possible because the total number of tiles loaded and rendered at any time is essentially bounded regardless of the zoom level. Figure 1 shows the visible tiles of a raster as solid colours to highlight the tile partitioning. The left side of the figure (showing the more zoomed-out view) exposes a larger area of the raster but conversely it can use tiles from a lower-resolution image in the pyramid. The right side of the figure (showing the more zoomed-in view) exposes a smaller area of the raster but conversely it needs tiles from a higher-resolution image in the pyramid. As can be seen in Fig. 1c this results in both zoom levels requiring roughly the same number of tiles and hence placing roughly the same load on the graphics hardware and streaming subsystems. The yellow wireframe in the figure is the view frustum used by the "3-D Orthographic" view in GPlates. A frustum, as typically used in computer graphics for a perspective projection, is the solid portion of a truncated pyramid whose top has been cut off by a plane parallel to its base (Neider et al., 1997) and where the pyramid apex represents the eye, or camera, location (the single point receiving the converging light rays). However, the frustum in an orthographic view is a rectangular prism (the light rays are parallel, instead of converging, and hence the eye/camera location is essentially at infinity). The yellow orthographic-view frustum in Fig. 1 does not include the rear hemisphere of the globe because raster data on the rear hemisphere are not normally visible through the globe (note that the globe is normally opaque grey in GPlates but is rendered semi-transparent in the figure in order to highlight the yellow wire-frame frustum within the interior of the globe).

2.4 Streaming raster data

As the user pans the view across the globe, new raster tiles come into view while old tiles drop from view. And as the user zooms the view into the globe, each visible tile is recursively replaced with four higher resolution tiles to enhance the detail and vice versa when zooming the view away from the globe. This continual acquiring and releasing of tiles necessitates the streaming of raster data from disk as the user interactively explores raster data sets.

GPlates uses the open-source, cross-platform graphics library OpenGL to access hardware-accelerated graphics functionality. For a raster tile to be rendered by the graphics processing unit (GPU), its data must first be loaded into an OpenGL texture resource. The tile texture dimension is fixed at 256×256 pixels since all graphics hardware supports this texture dimension and it provides a reasonable granularity for visibility culling. GPlates maintains a cache of texture resources in memory for each raster that includes the currently visible tiles plus the most recently released (not visible) tiles. When a new tile becomes visible GPlates first searches the texture cache. If the tile is not cached then the tile data is loaded from disk (the multi-resolution tiled image pyramid cache file) into the least recently released texture resource. The texture cache and the Operating System's physical-memory-backed file cache help to minimize the impact of high-latency disk accesses. For example, panning the view over a new area of a raster can initially result in some stuttering but when the user later returns to the same area the panning tends to be much smoother.

To further mitigate the cost of high-latency disk accesses, the raster tiles are not stored on disk as rows of tiles across the raster (storing the entire first row followed by the second, third, etc). Instead the tiles are arranged on disk in a Hilbert space-filling curve (Sagan, 1994) in order to minimize the file seek distance between spatially adjacent tiles. This is not very noticeable for low-resolution rasters, but becomes increasingly important for higher-resolution rasters. With the advent of low-latency SSDs this is not nearly as important as it is for the relatively high-latency HDDs.

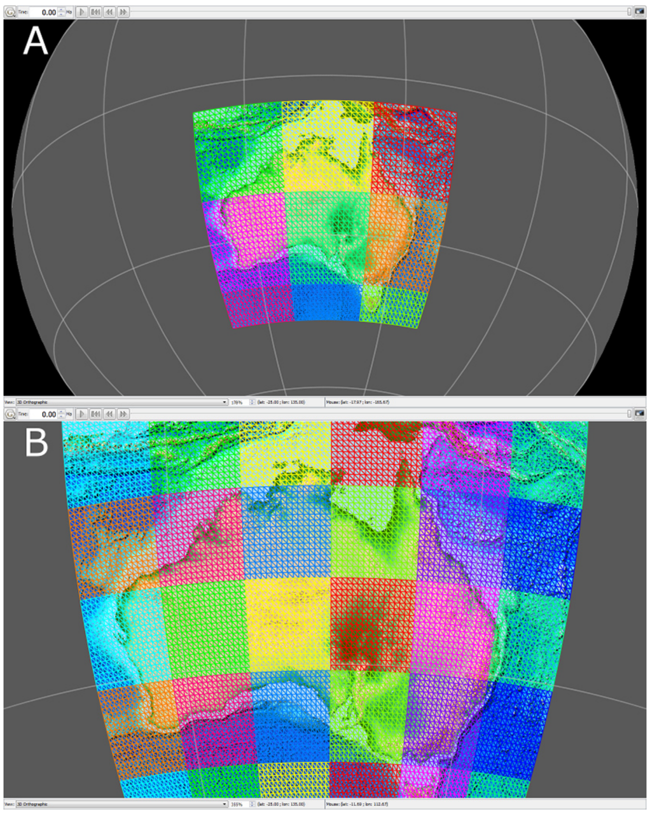

Figure 2. A regional (non-global) raster overlaid with colour-coded wireframe geo-reference tile meshes at two different tile resolutions (zoom levels). There are more tiles in **(B)** than in **(A)**, covering the same geographic region, because the next-higher resolution of the image pyramid (four times the pixels) is required for the zoomed-in view **(B)** compared to the zoomed-out view **(A)**.

2.5 Geo-referencing raster data

The pixels in a raster tile are positioned on the globe (geo-referenced) using either an affine transform or a sequence of ground control points. The geo-referenced coordinates are specified in the raster's coordinate reference system, which can include a map projection (such as Lambert Conic Conformal). The GDAL library (Warmerdam, 2008) is used to convert geo-referenced coordinates to the default coordinate reference system World Geodetic System 1984 (WGS 84) (Kumar, 1988) in GPlates.

GPlates uses OpenGL to render a raster tile by first creating a regular surface mesh containing geo-referenced vertices and attaching the tile's texture. Each vertex in the mesh contains a 3-D position on the globe and a 2-D texture coordinate (location in the tile texture). Since the geo-referencing cannot be calculated at each raster pixel it is only accurate at mesh vertices. To help minimize this visual deviation between vertices, adjacent vertices are separated by no more than 16 pixels. Therefore one raster tile (256×256 pixels) is represented by a mesh of 17×17 vertices (including tile boundary vertices). Figure 2 illustrates how the mesh spacing adapts to the level in the raster image pyramid such that the number of raster pixels between adjacent mesh vertices is roughly the same (in Fig. 2a and b).

2.6 Rendering raster data

Rendering a raster involves first determining the resolution level in the multi-resolution image pyramid and then finding the visible geo-referenced tiles in that level. To determine tile visibility, each tile is bounded by an oriented bounding box (OBB) that is tested for intersection with the view frustum. An OBB is a rectangular 3-D bounding box at an arbitrary orientation. To improve culling efficiency all tiles in each resolution level are arranged in an OBB binary tree so that, during rendering, hierarchical visibility culling can be employed to cull an entire group of tiles with a single OBB-frustum test. The visible raster tiles are then rendered into the view window or, if the raster is reconstructed, into the cube map.

2.7 Time-dependent rasters

A raster consists of a single raster image typically representing geospatial data observations on the Earth's present-day surface. GPlates can also create a time-dependent raster by importing a time sequence of raster image files and assigning geological ages to each image. GPlates renders a time-dependent raster by selecting the image, in its time sequence, whose age most closely matches the current geological time. Note that it is also possible to reconstruct a time-dependent raster. In this case, both the image and spatial location of the time-dependent raster vary with geological time. Raster reconstruction is covered in the next section.

3 Multi-resolution raster reconstruction

The rendering framework described so far can render an unreconstructed raster either directly to the computer screen when the user has chosen not to reconstruct a raster or as the first stage of the raster reconstruction process when the user does choose to reconstruct a raster.

This section describes our approach to visualization of a reconstructed present-day (or time-dependent) raster by building a multi-resolution cube map framework on top of the raster rendering framework described so far.

Our raster reconstruction process involves a multi-resolution cube map, a set of tectonic polygons and a rotation model. By imprinting the raster data into the overlapping tectonic polygons and then independently rotating the polygons across the globe, using the rotation model, we are essentially reconstructing present-day raster data to its past geological configuration. First we introduce the concept of a cube map as an efficient way to capture raster data in a representation that is decoupled from the raster's geo-referencing and in-built raster projection. We then extend the cube map with multi-resolution tiles to support level of detail and visibility culling necessary for efficient rendering. We show that any tile, in the cube map, can be generated by rendering the geo-referenced raster using the method described in Sect. 2. With the generation of a multi-resolution cube map covered, we finally describe in detail how raster data are imprinted onto tectonic polygons and rotated with them across the globe. The multi-resolution cube map facilitates this by enabling the raster imprinting of pre-rendered cube map tiles onto tectonic polygons to be accelerated by common graphics hardware.

3.1 Cube map properties

The cube map was first proposed by Ned Greene (Greene, 1986) as an alternative method of capturing the full projection of the world or environment from a particular world location. This is useful for applications such as reflection mapping because the cube map can be efficiently generated and indexed using common graphics hardware (compared to spherical and paraboloid mappings). An environment cube map is created by placing the cube centre at a particular world location and then projecting the environment onto the six faces of the cube. This is achieved by separately rendering the six views into six square textures each using a 90° field-of-view perspective frustum. The environment cube map then consists of these six textures arranged as a single hardware cube map texture (since the GeForce256 GPU was introduced in 1999) whereby a 3-D direction vector can be used to look up the cube map texture. For example, in the reflection mapping scenario the 3-D look-up vector is a surface reflection vector calculated at each point on the reflecting object's surface such that when the entire surface is rendered it will appear to reflect the environment.

For GPlates the cube centre is always positioned at the centre of the globe and the environment captured by the cube map is the raster data as it appears on the surface of the globe. Essentially we're looking from the centre of the globe at the inside of the globe as if the interior of the globe was empty and the raster was painted on the inside surface of the globe. The view direction of each of the six view frustums is aligned with a coordinate system axis ($+x$, $-x$, $+y$, $-y$, $+z$, $-z$). Figure 3 shows a global raster captured in an environment cube map.

Once the raster cube map is captured it can then be accessed using a 3-D vector to look up a raster pixel. The 3-D look-up vector is simply the direction from the centre of the globe to a point on the surface of the globe. For a unit sphere centred at the coordinate system origin this is the same as the 3-D position on the surface of the globe. The raster pixel retrieved from the cube map is the value of the geo-referenced raster at the specified surface position on the globe.

A cube map has a higher sampling density near the face corners compared with the face centres resulting in a non-uniform sampling of the cube map pixels across the surface

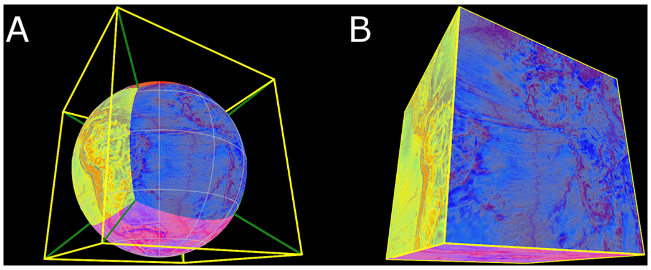

Figure 3. A global free-air gravity raster (Andersen et al., 2010) highlighted with solid colours representing the faces of a cube. Each cube face and the globe centre represent a perspective (pyramid) frustum. The yellow wireframe represents the cube and the green wireframe represents the edges of the six pyramid frusta. **(A)** Coloured raster regions captured by the pyramid frusta of the cube faces; **(B)** coloured raster regions radially projected onto the cube faces (from the globe centre).

of the globe. However, the non-uniformity is much less than the common rectangular (latitude–longitude) projection and avoids the projection singularity at the North and South Poles.

3.2 A multi-resolution cube map

Essentially a cube map raster and a geo-referenced raster are different representations of the same raster data. So, as with the geo-referenced raster, the cube map raster is extended to include multi-resolution capabilities. That is, to enable matching the cube map raster resolution to the display (or render target) resolution and to enable visibility culling by partitioning each face of the cube into tiles.

A multi-resolution cube map hierarchically partitions each face of a cube into tiles using a quadtree data structure (Finkel and Bentley, 1974) whereby the 2-D surface of each cube face is divided into four quadrants and each quadrant further divided into four sub-quadrants and so on. Each node in the quadtree represents a fixed-dimension tile. Each level, or depth, in the quadtree represents a different resolution of the cube map raster since each level contains, relative to its parent level, four times the number of tiles (and hence four times the pixels). For example if the tile dimension is 256 then quadtree level 0 is one tile (or 256×256 pixels), level 1 is 2×2 tiles (or 512×512 pixels), level 2 is 4×4 tiles (or 1024×1024 pixels), etc. Figure 4 shows the association between the quadtree data structure, the hierarchy of tile images and the tile frustums within a single cube face. Also note that there are six quadtrees in total since each cube face has its own quad tree.

3.3 Generating a cube map raster

A cube map raster is generated by rendering a geo-referenced raster into it. This uses the same rendering procedure that was used to render a geo-referenced raster into the view window, as described in Sect. 2, with the exception that the view frustum and target frame buffer differ for the cube map. Each tile in a multi-resolution cube map has its own view frustum and target texture resource. A cube map tile texture is generated by rendering only those geo-referenced tiles that are visible in its view frustum. During rendering, the geo-referenced tile vertices are transformed by the 4×4 off-axis perspective (pyramid) projection transformation (Neider et al., 1997) defined by the frustum of that cube map tile. Figure 5 shows the geo-referenced tiles, as solid colours, visible in the view frustum of a single cube map tile.

A cube map raster contains all such cube map tiles whose frusta intersect the geo-referenced raster (for a global raster, this includes all cube map tiles). Figure 6 shows these cube map tiles for a regional geo-referenced raster at three different resolution levels. Only those cube map tiles that actually cover the raster's region are required. Note that the solid colours in Fig. 6 highlight the cube map tiles as opposed to Fig. 5 which highlights the geo-referenced tiles (for a single cube map tile).

For graphics hardware supporting non-power-of-two texture dimensions the tile dimension of the cube map is optimally adapted for each geo-referenced raster in order to minimize texture memory usage.

3.4 Rendering reconstructed raster data

Raster data are reconstructed by combining a raster cube map, a tectonic polygon data set and a rotation model. A tectonic polygon represents the boundary of a tectonic plate – a region of the Earth's crust that is typically rigid throughout geological time. The rotation model determines the motion of each tectonic plate across the globe over geological time. Each tectonic polygon is linked to the rotation model using a tectonic plate ID which provides access to a time sequence of finite Euler rotations (Boyden et al., 2011; Greiner, 1999) that rotate the tectonic plate from its present-day position to its position at a time in the geological past relative to another tectonic plate (with a different plate ID). These plate-relative rotations are then accumulated by traversing the plate circuit from the tectonic plate back to the global reference frame to obtain the absolute rotation of the tectonic plate (Cox and Hart, 1986). Any geospatial data, including rasters, attached to a tectonic plate will inherit that plate's motion over geological time. Vector geometry such as points, polylines and polygons can be reconstructed once they are partitioned into tectonic plates (and assigned associated plate IDs). Raster data is conceptually reconstructed in a similar manner whereby raster pixels are attached to (or partitioned into) tectonic plates and then rotated using the motion of the attached plates. In GPlates the user attaches raster data to tectonic plates by connecting a polygon layer to a raster layer. Although the tectonic plates can rotate across the globe, their polygon shapes are static (no change over time). While GPlates does support dynamic plate polygons

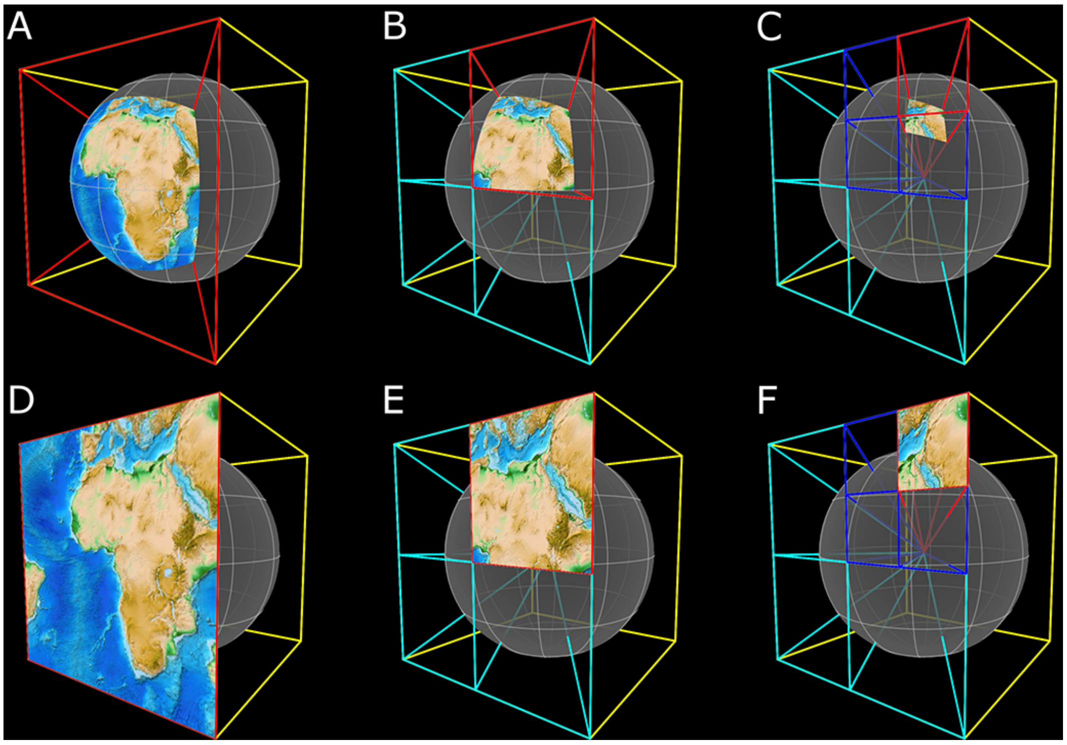

Figure 4. Quadtree partitioning of a single cube face of the ETOPO1 global raster (Amante and Eakins, 2009). The yellow wireframe represents the cube. The red wireframe represents the perspective (pyramid) frustum containing the highlighted raster tile – note that these can be off-axis pyramids (in **B, C, E** and **F**). **(A)** The tile at level 0 of the quadtree covering the entire cube face. **(B)** The four tiles at quadtree level 1 (cyan wireframe). These are the children of the tile in **(A)**. **(C)** Four (out of sixteen) tiles at quadtree level 2 (blue wireframe). These are the children of the tile rendered in **(B)**. **(D–F)** Same as **(A–C)** except that the rendered tiles are radially projected onto the cube face to show the actual tile textures (square 2-D images).

(Gurnis et al., 2012) to handle the changing shape of oceanic plates over geological time, they are not currently used to reconstruct rasters. A future extension of GPlates functionality will be designed to use dynamic plate polygons, along with deforming plates, to reconstruct and deform rasters.

A raster is efficiently reconstructed by mapping the tile textures of the raster cube map onto the tectonic plate polygons and compositing the rotated polygon-tile intersections into the scene. The level of detail selected for rendering is the lowest resolution of the raster cube map that maintains a crisp visual appearance of the reconstructed raster. Figure 7 shows the pseudocode for rendering a reconstructed raster.

The function render_reconstructed_raster iterates over the six faces of the raster cube map and, for each tectonic polygon, renders those cube map tiles (at the appropriate level of detail) that overlap the polygon. Each face of the cube map contains a quadtree hierarchy of tiles that, beginning with the root tile of a face, is recursively visited using the function render_raster_tile. Each level of recursion visits one level deeper in the quadtree and expands the number of tiles visited by up to four. If a visited tile is at the appropriate level of the quadtree for rendering, then the tile is mapped onto the polygon and then rotated and composited into the scene. Otherwise the four child tiles at the next deeper level are visited recursively. Tiles that do not overlap the polygon are culled, and rotated tiles that are not visible are also culled.

Figure 8 shows the accumulated rendering of a single rotated polygon mesh. The rendering level of detail is quadtree level 1. At this detail level there are three tiles overlapping the polygon. Those same three tiles are also shown in their present-day locations in Fig. 6b.

This process is repeated for all tectonic polygons until the reconstructed raster is fully composed into the scene. Figure 9 shows a raster reconstructed with two tectonic plates.

The following sections cover the most important details in this process.

3.4.1 Generating tectonic polygon triangulation meshes

Each tectonic polygon is rendered as a spherical mesh that is triangulated within the interior surface region of the polygon. The Computational Geometry Algorithms Library (CGAL) (Fabri and Pion, 2009) is used to generate a surface polygon mesh of sufficient tessellation to give a spherical appearance when rendering into the “3-D Orthographic” view in GPlates. This is a time-consuming process and as such it only occurs

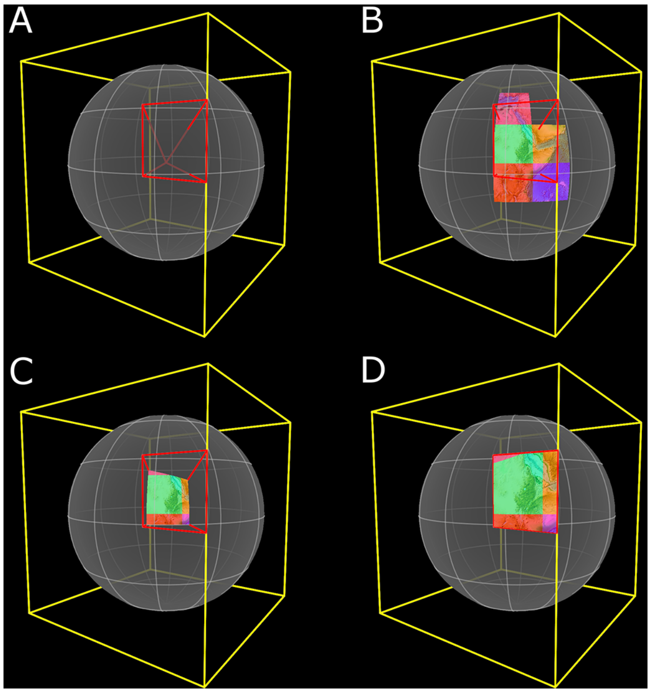

Figure 5. Rendering geo-referenced tiles into a single cube map tile of the ETOPO1 global raster (Amante and Eakins, 2009). The geo-referenced tiles are highlighted with different colours. The yellow wireframe represents the cube. **(A)** The red wireframe represents the perspective, off-axis pyramid frustum of the single cube tile. The pyramid apex is at the globe centre. **(B)** The subset of geo-referenced tiles that overlap the frustum of the single cube tile – note that tiles outside the frustum are culled. **(C)** The cube tile rendered on the sphere. This is the intersection of the geo-referenced tiles and the cube tile frustum. **(D)** The cube tile projected onto the cube face to show the actual cube tile texture (square 2-D image) and the contributions from the geo-referenced tiles.

when a raster layer is first connected to a polygon layer in GPlates. Figure 8 (b, d, f) shows the wireframe of continental Australia. Since the cube map decouples geo-referenced raster rendering (Sect. 2) from tectonic polygon rendering, the same tectonic polygon mesh data set can be shared by all raster layers attached to it.

3.4.2 Mapping a raster cube map onto polygon meshes

In the pseudocode in Fig. 7, each atomic rendering operation submitted to OpenGL consists of a rotated polygon mesh rendered with a cube map tile texture. In most applications a texture is usually mapped onto a mesh by assigning a 2-D texture coordinate to each mesh vertex. An advantage of the cube map approach is that the tectonic polygon meshes can be rendered without requiring explicit texture coordinates. Instead, the surface positions in the mesh vertices are transformed into texture coordinates directly by the graphics hardware during polygon mesh rendering. Automatic texture coordinate generation dates back to base OpenGL 1.1 and supports matrix transformation of vertex positions for use as texture coordinates. GPlates uses this functionality to project the 3-D position of each polygon mesh vertex into the 2-D texture space of a cube map tile. This 4×4 texture matrix transformation is identical to the matrix used to render the geo-referenced raster into the cube map tile texture. This has the same effect as unprojecting the pre-rendered cube map tile texture back onto the globe during rendering of the surface polygon mesh. Note that the graphics hardware uses perspective-correct texture coordinate interpolation when rasterizing pixels (of the polygon mesh) and hence there is no texture projection deviation or mismatch between generating a cube map tile texture (Sect. 3.3) and unprojecting it back onto the globe.

In addition to transforming the 3-D positions of mesh vertices to generate 2-D texture coordinates, the 3-D positions are transformed to generate the final rotated polygon mesh positions on the surface of the globe using the Euler rotation of the tectonic polygon. The combined effect of these two transformations is to project a raster onto a polygon mesh and then rotate it across the globe. Both 4×4 transformations are accelerated by the graphics hardware.

3.4.3 Clipping polygon meshes to cube map tiles

Graphics hardware can apply a limited number of textures when rendering a mesh (with older hardware supporting a maximum of four textures). Each cube map tile is rendered using up to four textures in some cases, including the source tile texture itself, the clip texture (see below) and an optional age grid mask and age grid coverage texture (age grids are described in Sect. 4.3). So even though a single polygon mesh can overlap multiple cube map tiles, it must be rendered separately for each tile. In this case only the intersection of a polygon mesh with each cube map tile can be rendered. This is achieved by rendering the entire polygon mesh for each cube map tile but only rasterizing pixels that map to a tile's interior. Pixels that map to a tile's exterior are rejected by modulating the tile's texture with a clip texture and using the Alpha Test functionality of OpenGL 1.1 to discard transparent pixels (alpha equal to zero). The clip texture is a small 4×4 pixel texture containing opaque inner pixels and transparent outer pixels. The boundary around the inner 2×2 pixels maps to the tile boundary such that polygon mesh regions outside the tile become transparent. Figure 8 (b, d, f) illustrates the clipping of a single rotated polygon mesh by the three tiles that overlap it.

3.4.4 Intersections between cube map tiles and tectonic polygons

The pseudocode in Fig. 7 includes a test for the intersection of a present-day (unrotated) tectonic polygon with an unrotated cube map tile. These intersections are independent of the geological time of reconstruction since they do not

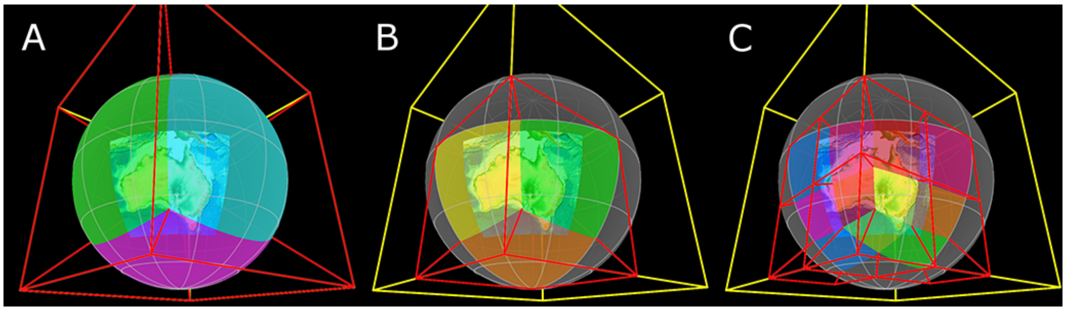

Figure 6. The cube map tiles (highlighted in different colours) of a regional raster at three different resolution levels. Since the raster is regional, not all cube map tiles are required – only those that actually cover the raster's region. This highlights the partial nature of quadtrees of regional rasters. **(A)** Only three cube faces (and hence three quadtrees) are required to provide coverage of the regional raster at quadtree level 0. **(B)** For each of the three cube face tiles in **(A)** only one of four possible child tiles (quadtree level 1) are required to cover the raster. **(C)** The cube map tiles at quadtree level 2 (children of the tiles in **B**) that cover the raster.

```
render_reconstructed_raster(raster, tectonic polygons)
   for each face in raster cube map
        if raster covers cube face then
             for each polygon in tectonic polygons
                  render_raster_tile(root tile of face, polygon)

render_raster_tile(tile, polygon)
   if polygon’s present-day geometry intersects tile then
        if rotated tile bounding box is visible then
             if reached level-of-detail to render then
                  render rotated polygon clipped to tile texture
             else
                  for each child tile in current tile
                     if raster covers child tile then
                          render_raster_tile(child tile, polygon)
```

Figure 7. Pseudocode for the rendering traversal of a reconstructed multi-resolution cube map raster.

involve rotations and hence their boolean results can be pre-computed for all tile–polygon pairs. And since the tile partitioning spatial configuration is the same for all raster cube maps, regardless of their individual geo-referencing, the intersection results can be shared by all rasters that are attached to the same tectonic polygon data set.

3.4.5 Visibility culling of cube map tiles

The pseudocode in Fig. 7 includes a test to determine if a rotated tile is visible in the scene's view frustum when rendering a tectonic polygon. Each cube map tile is bounded by an OBB in much the same way as geo-referenced tiles are bounded (Sect. 2.6). Visibility culling involves rotating the OBB of each cube map tile that overlaps a tectonic polygon (using the Euler rotation of the tectonic polygon). A rotated tile is then only rendered if its rotated OBB intersects the scene's view frustum. Figure 8 (a, c, e) shows the rotated tile OBBs in red wireframe and the scene's view frustum in yellow wireframe. As with geo-referenced tiles the visibility culling is hierarchical since a single OBB-Frustum test can cull an entire sub-tree of quadtree tiles. Also, it turns out that, instead of rotating each tile OBB, it is actually more efficient to reverse-rotate the scene view frustum once (per tectonic polygon) and hierarchically test against the unrotated OBBs. A single polygon typically has multiple overlapping tiles. Conversely a single tile can be covered by multiple (adjacent) polygons.

3.4.6 Removing texture seams between cube map tiles

Cube map tile textures are rendered using hardware bilinear filtering to reduce the texture sampling artifacts of nearest-neighbour filtering. However, this introduces visible raster seams or discontinuities across tile boundaries because the filtered texture samples of two adjacent tiles do not match along the shared tile boundary. To remove these seams each cube map tile frustum is expanded by half a pixel such that adjacent tile frusta overlap slightly. Note that this does not introduce a half-pixel error because the geo-referenced raster is also rendered into the adjusted frusta when generating the cube map tiles.

Since GPlates adapts the raster cube map resolution to the display (or render target), hardware texture mipmaps (Williams, 1983) are not required to reduce frequency-aliasing artifacts. This also avoids the problem of the half pixel frusta overlap solution not always working with hardware texture mipmaps (due to pixel size varying across mipmap levels). However, hardware anisotropic texture filtering is used to reduce aliasing in highly anisotropic regions of the globe where the surface normal is almost perpendicular to the view direction (such as near the horizon).

4 Raster reconstruction enhancements

The rendering framework described so far can render a reconstructed raster directly to the computer screen in the standard 3-D globe view. This section describes various enhancements to raster reconstruction that build on the approach described in Sect. 3.

Visualizing reconstructed rasters in the 2-D map projection views discusses how interactivity is maintained in the

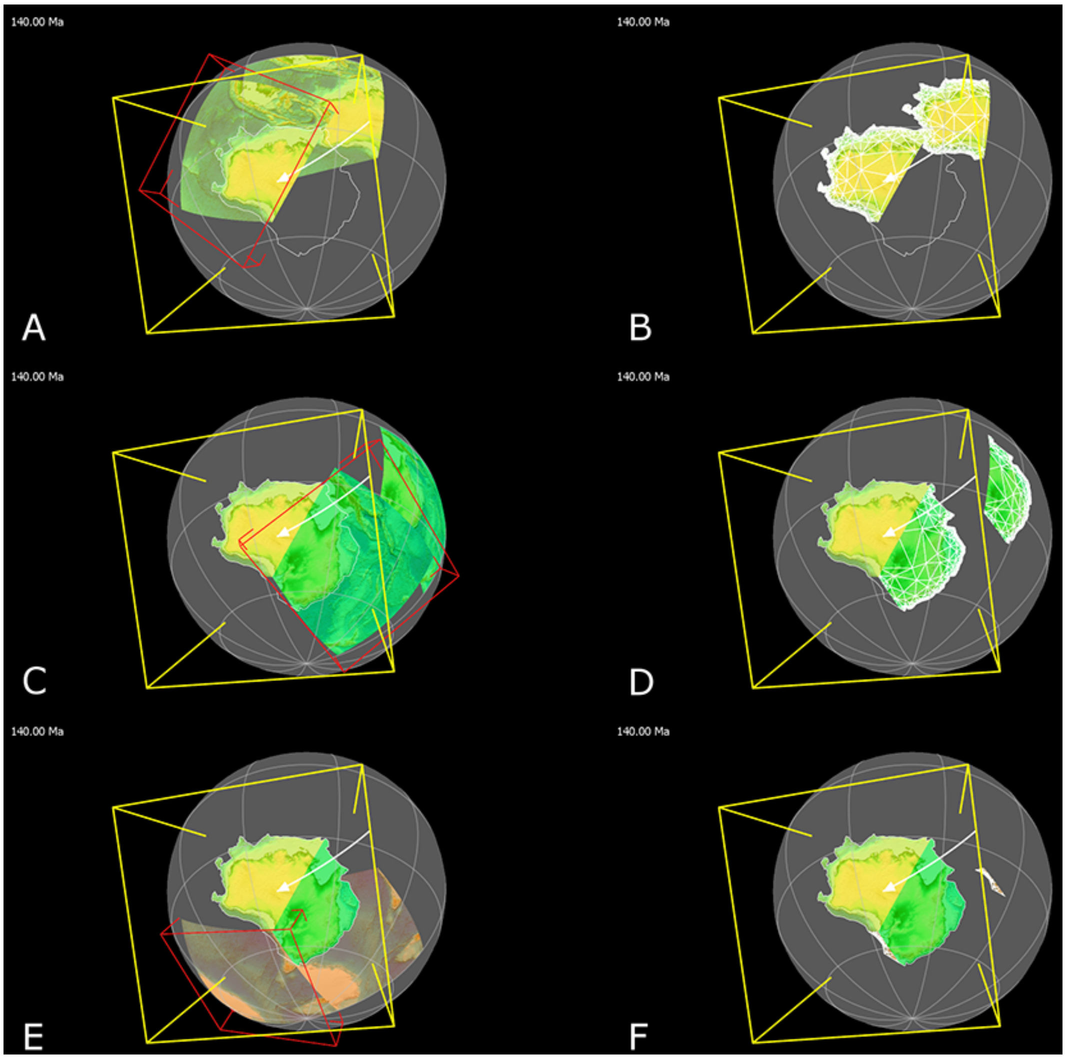

Figure 8. Accumulated tile rendering of single rotated polygon mesh (continental crust on the Australian plate) overlapping three cube map raster tiles (at quadtree level 1). The rotation (illustrated by the white curved arrow) is from present day (0 Ma) to 140 Ma. The yellow wireframe is the orthographic-(rectangular prism) view frustum used by the "3-D Orthographic" view in GPlates. The three tiles are also shown in Fig. 6b in their present-day locations. **(A, C, E)** show the unrotated and rotated cube map tiles. Also shown are the rotated tile OBBs in red wireframe. All three OBBs intersect the orthographic-view frustum and therefore are visible; **(B, D, F)** show the unrotated and rotated polygon mesh, in white wireframe, clipped to each tile.

2-D map views. Instead of rendering the reconstructed raster directly to the computer screen (as done in the 3-D globe view), it is first rendered into an intermediate cube map (called the reconstructed cube map) and then rendered directly to the computer screen via a map projection.

Improved raster reconstruction using age grids makes use of a second raster, containing pixel values representing the age of oceanic crust. Replacing the usual per-tectonic-polygon age test with a per-pixel age test restores the missing sections of reconstructed raster data near oceanic ridges.

Revealing raster detail with surface normal maps modulates the colour of a raster with the surface lighting generated from a second raster. This enables patterns in the second raster to be visually transposed onto the first raster.

And finally, analysing reconstructed numerical raster data adds support for reconstructing floating-point raster data (in contrast to colour raster data). This enables numerical raster data to be analysed in the data-mining front-end of GPlates and also enables export to numerical raster file formats.

4.1 Visualizing reconstructed rasters in 2-D map projection views

The default view in GPlates is the 3-D Orthographic Globe view that uses an orthographic-view projection commonly used in computer-aided design (CAD) software where the lines of projection are parallel. GPlates also supports a variety of 2-D map projection views including the rectangular, Mercator, Mollweide and Robinson projections that unwrap the globe onto a 2-D plane.

Rendering reconstructed raster data to a 2-D map view requires an extra rendering phase above that required for the 3-D globe view. Instead of rendering directly to the computer screen (as done for the 3-D globe view) the reconstructed raster is rendered into a reconstructed cube map which is, in turn, rendered into the 2-D map view. This also serves to demonstrate that a cube map can capture any surface representation on the globe. The cube map in Sect. 3.3 captures the unreconstructed, geo-referenced raster. Here the reconstructed cube map captures the final reconstructed raster (the output of Sect.3.4). The 3-D globe view does not need a reconstructed cube map because the 3-D orthographic-view projection and each tectonic polygon Euler rotation can be combined into a single 4×4 matrix that can be accelerated by graphics hardware dating back to OpenGL 1.1. For the 2-D map views the surface positions on the globe are converted into 2-D map coordinates using the Proj4 library (Evenden and Warmerdam, 1990) and this library cannot be offloaded to the graphics hardware. Even if that were possible the graphics hardware would still need to clip the rotated tectonic polygons across the 2-D map boundary and this is not something that most graphics hardware can do. While this could all be done on the central processing unit (CPU), it would be significantly slower. Instead the alternate solution of rendering to a reconstructed cube map keeps the 2-D map views interactive by decoupling raster reconstruction from map projection. This removes the need to perform expensive per-vertex map projections for each rendered frame and removes the need to clip rotated tectonic polygon meshes to the 2-D map boundary (the reconstructed cube map is aligned with the central meridian of the map projection). The contents of the reconstructed cube map are rendered to the 2-D map view using a static latitude–longitude mesh with each vertex containing a 2-D map view position and its associated 3-D position on the globe. To access the reconstructed raster data the graphics hardware projects the 3-D position, in each vertex, into a reconstructed cube map tile texture. The mesh tessellation is dense enough to provide a visually accurate sampling of the map projection. And tile LOD determination and visibility culling for a 2-D map view is done using the hierarchical partitioning of the reconstructed cube map (in a similar manner to that described in Sect. 3).

Figure 10 shows a raster reconstructed with two tectonic plates in a 2-D map projection view (similar to Fig. 9, which instead uses the 3-D orthographic view). Note that, unlike Fig. 9c, the cube map tiles in Fig. 10c do not retain their shape after rotation. This is due to the 2-D map projection and is particularly noticeable with Antarctica.

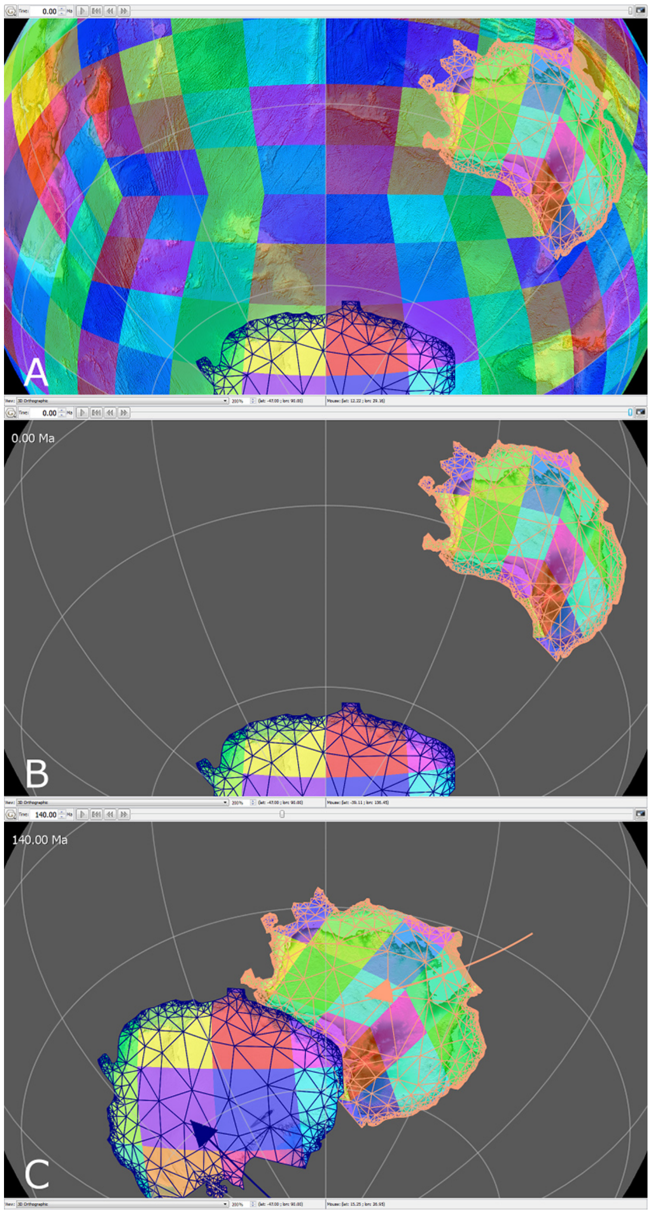

Figure 9. The GPlates "3-D Orthographic" view of the ETOPO1 global raster (Amante and Eakins, 2009) reconstructed to 140 Ma using the Australian and Antarctic continents. The cube map tiles are highlighted in different colours. **(A)** The entire unreconstructed raster with the tectonic polygon meshes overlaid on top. Note the cube-like arrangement of tiles – two cube corners are noticeable in the pattern of tiles. **(B)** The tectonic polygon meshes in their present-day (0 Ma) locations essentially cut out polygon-shaped blocks in the global raster and its tiles. **(C)** The tectonic polygon meshes in their reconstructed locations at 140 Ma. Each plate rotation is illustrated by a curved arrow. The raster and its tiles rotate independently with each tectonic polygon mesh.

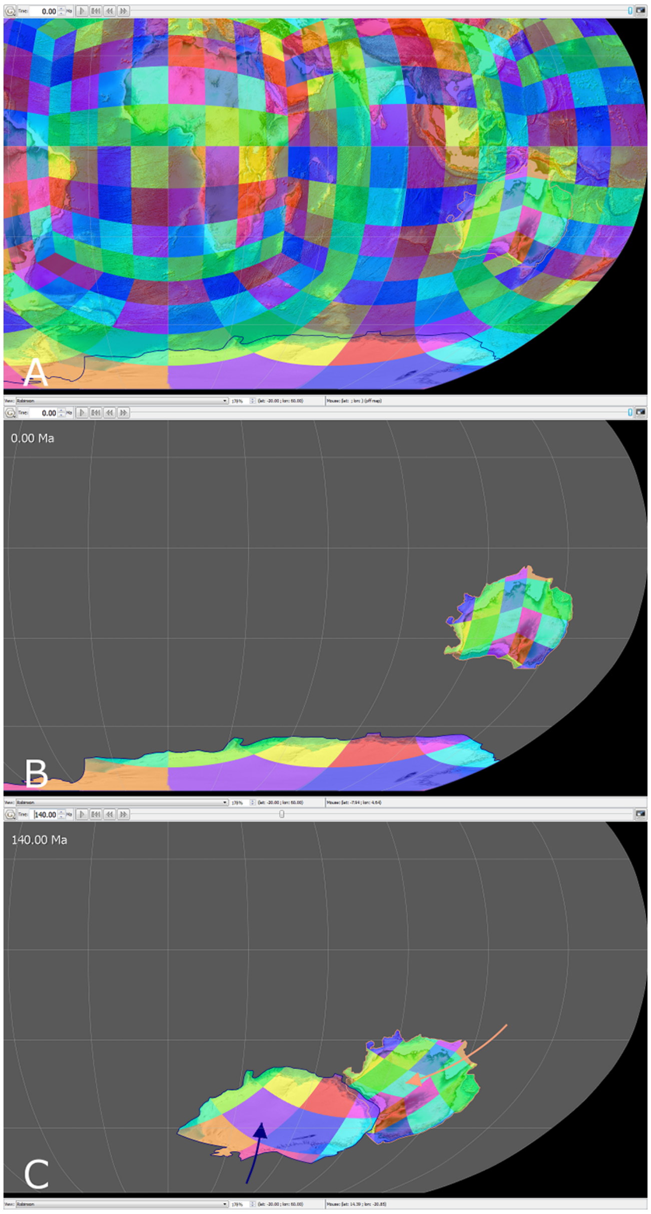

Figure 10. The GPlates "Robinson" 2-D map projection view of the ETOPO1 global raster (Amante and Eakins, 2009), reconstructed to 140 Ma for continental Australia and Antarctica. The cube map tiles are highlighted in different colours. This is the equivalent of Fig. 9 but with a 2-D map projection view instead of the 3-D orthographic view. **(A)** The entire unreconstructed raster with raster tiles following a cube-like arrangement. **(B)** The tectonic polygons in their present-day (0 Ma) locations. **(C)** The tectonic polygons in their reconstructed locations at 140 Ma. Each plate rotation is illustrated by a curved arrow.

4.2 Combining multiple rasters

So far we have dealt with a single reconstructed source raster. In Sects. 4.3 and 4.4, new rasters are introduced that are combined with the source raster in different ways. However, two rasters cannot easily be combined if they have different geo-referencing or different raster inbuilt map projections. This is because the geo-referenced tiles of two different rasters will not, in general, be aligned across the two rasters or projected in the same way and hence cannot be paired or associated. But since all rasters have the same cube map spatial partitioning, regardless of their individual geo-referencing, they can be combined after their cube maps have been generated. This procedure is used for age grids and surface normal maps in the following sections.

4.3 Enhancing raster reconstruction with a crustal age grid

Each tectonic polygon has a time of appearance and disappearance corresponding to the geological time period over which the crust, contained within the interior of the polygon, existed on the Earth's surface. The pseudocode in Fig. 7 renders only the subset of tectonic polygons that exist at a particular reconstruction time. In order to simulate the gradual generation of oceanic crust at mid-ocean ridges, many long, thin oceanic polygons with varying times of appearance have been digitized along these ridges. These oceanic polygons progressively appear as GPlates animates the reconstruction time forward in time. However, the granularity of this transition is still quite coarse, with regions (the size of polygons) continually popping into view. A finer per-pixel granularity is achieved by using an age grid raster.

The age grid is a separate raster that contains the time of appearance in each pixel (Müller et al., 2008). It follows the same principle as polygons but is reduced in spatial extent to pixel-size regions of crust. The tectonic polygons are still used during raster reconstruction, but now the per-polygon age comparison is replaced with a per-pixel age comparison, effectively removing the polygon popping. This per-pixel comparison is efficiently performed using the graphics hardware. For OpenGL 2.0 hardware this test is simply a floating-point comparison in an OpenGL Shading Language (OpenGL Architecture Review Board, 2014) shader program. For older hardware some rendering trickery using fixed-point hardware and four texture units is required to simulate this comparison. Figure 11 shows the reconstruction of a raster with and without the assistance of an age grid.

4.4 Enhancing visual detail with surface normal maps

A raster can be visually transposed onto another raster by combining the colour of the first raster with the surface lighting of the second. The second raster is treated like a height-field from which surface relief shading is generated. The colour of the first raster is then modulated with the surface lighting (intensity) of the second raster to get the final output colour. Figure 12 shows two rasters that source lighting from themselves and from each other. Figure 12d uses the age grid for surface relief (height-field) – the regions near the

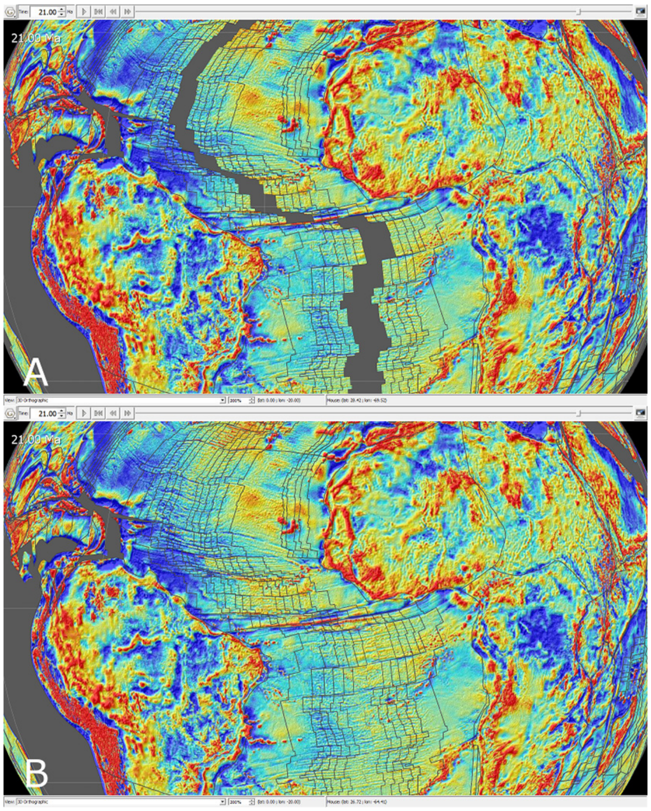

Figure 11. The global free-air gravity raster (Andersen et al., 2010), reconstructed to 21 Ma. Overlaid on top is the static polygon data set (Seton et al., 2012) used to reconstruct the raster. Note the many long, thin oceanic polygons. **(A)** Reconstructed without using an age grid. The gap between Africa and South America is a result of the per-polygon age test. **(B)** Reconstructed using an age grid. The gap is no longer present because the age test is now per-pixel.

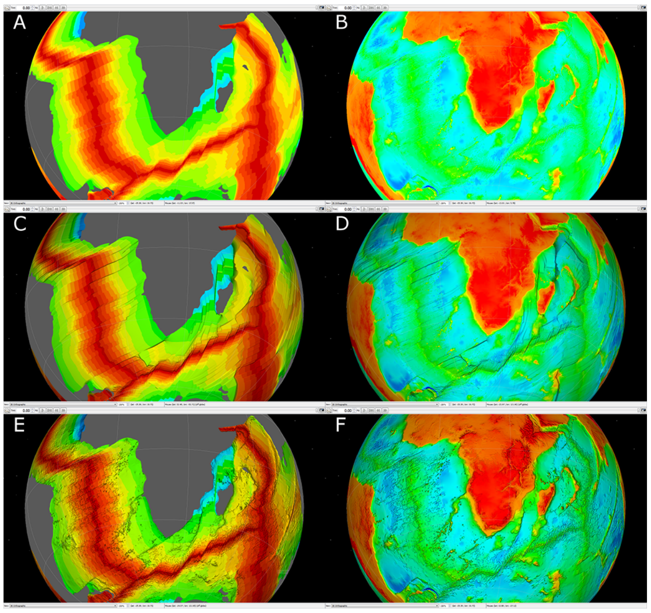

Figure 12. The left side shows the oceanic crustal age grid raster (Müller et al., 2008). The right side shows the General Bathymetric Chart of the Oceans (GEBCO) raster (Goodwillie, 2003). **(A)** age grid colour only; **(B)** GEBCO colour only; **(C)** age grid colour with age grid relief; **(D)** GEBCO colour with age grid relief; **(E)** age grid colour with GEBCO relief; **(F)** GEBCO colour with GEBCO relief.

mid-ocean ridge (between Africa and South America) appear to have a lower elevation than surrounding regions due to the lower age grid values.

Surface lighting is recalculated and combined as the raster is reconstructed to geological times. In other words, surface relief shading is not prebaked into the raster but calculated interactively as diffuse lighting using the light direction and rotated surface normals. The surface normal tiles are generated on the fly from raw numerical (height) raster data as an extension of the streaming process (Sect. 2.4). This is done on the GPU to enhance performance when floating-point textures and OpenGL Shading Language (OpenGL Architecture Review Board, 2014) are supported. However, these generated surface normals are in the tangent space of the geo-referencing (aligned with the raster). When the geo-referenced surface normal raster is rendered into a cube map, the surface normals are converted to world space (aligned with the global Cartesian coordinate system). This enables a surface lighting cube map to be combined with another raster cube map (since the raster-specific geo-referencing dependency has been removed).

4.5 Analyzing reconstructed numerical raster data

The GPU is typically a separate processor with its own high-speed memory for storing working data (such as textures and rendered images). GPlates visualizes reconstructed numerical (floating-point) raster data by first translating it to colour data before uploading to the GPU as 8-bit-per-channel RGBA textures (supported by all graphics hardware). The reconstructed RGBA raster data is then scanned straight from GPU memory to the display screen. In this case the original (untranslated) floating-point raster data is not reconstructed. This is fine for visualization; however, there are situations where GPlates needs to access the reconstructed floating-point raster data itself.

Reconstructed raw numerical raster data is needed when GPlates exports to numerical image file formats (such as NetCDF) for analysis by other software. It is also needed in the data-mining front-end of GPlates (Landgrebe and Muller, 2011) where raster data is co-registered with seed geometries by applying operations (such as mean and standard deviation) to the reconstructed numerical raster pixels within a region of interest of the seed geometries (the details of which are beyond the scope of this article).

GPlates can reconstruct raw numerical raster data (as opposed to RGBA raster data), provided the graphics hardware has support for floating-point textures. In this case numerical raster data is first uploaded to the GPU as two-channel floating-point textures. One channel is used for the floating-point raster data and the other is for the raster coverage or opacity. The numerical raster data is then reconstructed on the GPU and rendered into floating-point target textures. This reconstructed raster data is then asynchronously downloaded from the GPU using pixel buffers (GL_ARB_pixel_buffer_object OpenGL extension) to avoid stalling the CPU, thereby allowing it to keep the GPU busy with subsequent reconstruction tasks.

5 OpenGL in GPlates

GPlates can visualize reconstructed raster data on any graphics hardware manufactured in the last decade. This is because raster reconstruction visualization requires only base OpenGL 1.1 functionality plus support for destination alpha (for writing raster opacity to target textures) and four texture units; this is well supported by all graphics hardware from that time period.

However, some remote desktop software (where GPlates runs on the remote system) lacks support for hardware-accelerated graphics and falls back on OpenGL software rendering. This can be problematic if the software rendering is either too slow or lacks support for destination alpha and four texture units. An example of the latter case is Microsoft's OpenGL 1.1 software renderer, written in 1996 (and essentially unchanged since).

In general GPlates takes advantage of more advanced graphics functionality when it is available but does not require it. For example the OpenGL Shading Language (GLSL) (OpenGL Architecture Review Board, 2014), based on the syntax of the C programming language, affords GPlates more flexibility and performance in rendering and also enables normal map surface lighting. GPlates also makes heavy use of render-to-texture during raster reconstruction, and the well-supported GL_EXT_framebuffer_object capability improves performance by enabling direct rendering to textures – with GPlates falling back on indirect rendering, via the main frame buffer, when this is not supported.

Reconstructing raw numerical (floating-point) raster data for non-visualization purposes such as raster export or analysis requires graphics hardware supporting the GL_ARB_texture_float and GL_ARB_pixel_buffer_object OpenGL capabilities as well as GLSL (roughly equivalent to the functionality of OpenGL version 2.1), which, while not as ubiquitously supported as OpenGL 1.1, is still well supported by most graphics hardware in use today.

6 Discussion

The main advantage of cube maps in a GPlates context is a simplified implementation of raster reconstruction. The cube map approach decouples the rendering of the geo-referenced raster from its reconstruction, thereby avoiding the need to find the spatial intersections of tectonic regions within the hierarchy of geo-referenced tile regions, which would, in turn, require polygon-on-the-sphere intersection and triangulation algorithms that are numerically robust in the presence of self-intersecting polygons (with interior holes).

Another advantage of cube map rasters is the ability to combine multiple rasters that have different geo-referencing, such as enhancing reconstruction with an age grid (Fig. 11) and visually transposing two rasters using surface relief lighting (Fig. 12).

Although only briefly mentioned before, a cube map raster also enables efficient co-registration of a raster with seed geometries (points, polylines and polygons). This is because the seed geometries are inserted into a spatial partition that follows a hierarchical cube map structure analogous to that of the raster's cube map. This enables spatial associations between the raster and seed geometries to be quickly found.

A potential disadvantage of the cube map approach is that the intermediate cube map generation phase results in an additional resampling stage that can reduce the quality of the raster data due to the bilinear filtering. For the 2-D map views, there is a further cube map generation phase (Sect. 4.1), resulting in another resampling stage.

Another disadvantage is the non-uniform sampling of the cube map pixels that varies by a factor of approximately two, or more accurately 3/sqrt(2), from a cube face centre to one of its corners. For some cube map tiles this can result in up to four (2×2) times as many pixels rendered than is necessary for no loss of visual detail. This disadvantage does lessen for the higher-resolution LODs (since those tiles cover less area of each cube face). However, the flexibility gained by having a raster in cube map form is very advantageous.

7 Conclusions

This paper describes a method of reconstructing raster data that integrates a cube environment map with multi-resolution tile partitioning. This enables on-demand tile streaming, level-of-detail rendering and hierarchical visibility culling to be applied to rasters reconstructed by tectonic plate polygons. Furthermore the partitioned cube map tiles require only 4×4 matrix transformations which are extremely well supported by graphics hardware. With this implementation GPlates can reconstruct raster data on a wide variety of desktop and laptop computers manufactured over the last decade, enabling users to interactively visualize and analyse arbitrarily high-resolution geophysical raster data for geological times in the past. This capability forms the basis for interactively building and improving plate reconstructions in an iterative fashion, for tectonically complex regions as well as for ancient supercontinent assemblies, which can easily be analysed using GPlates by assigning geophysical grids and geological data to individual tectonic elements and interactively testing their alignment in alternative reconstructions (Williams et al., 2012).

Acknowledgements. R. D. Müller and J. Cannon were supported by ARC grant FL0992245, and J. Cannon and E. Lau were also supported by the AuScope National Collaborative Research Infrastructure (www.auscope.org.au/).

Edited by: J. C. Afonso

References

Amante, C. and Eakins, B. W.: ETOPO1 1 Arc-Minute Global Relief Model: Procedures, Data Sources and Analysis, US Department of Commerce, National Oceanic and Atmospheric Administration, National Environmental Satellite, Data, and Information Service, National Geophysical Data Center, Marine Geology and Geophysics Division, 2009.

Andersen, O. B., Knudsen, P., and Berry, P. A.: The DNSC08GRA global marine gravity field from double retracked satellite altimetry, J. Geodesy., 84, 191–199, 2010.

Boyden, J. A., Müller, R. D., Gurnis, M., Torsvik, T. H., Clark, J. A., Turner, M., Ivey-Law, H., Watson, R. J., and Cannon, J. S.: Next-generation plate-tectonic reconstructions using GPlates, in: Geoinformatics: Cyberinfrastructure for the Solid Earth, edited by: Keller, G. R. and Baru, C., Cambridge University Press, Cambridge, 95–114, 2011.

Cox, A. and Hart, B. R.: Plate Tectonics: How It Works, Blackwell Science Inc., 400 pp., 1986.

Fabri, A. and Pion, S.: CGAL: the Computational Geometry Algorithms Library, in: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, Washington, 538–539, 2009.

Finkel, R. A. and Bentley, J. L.: Quad trees a data structure for retrieval on composite keys, Acta Inform., 4, 1–9, doi:10.1007/BF00288933, 1974.

Goodwillie, A.: User Guide to the GEBCO One Minute Grid, Centenary Edition of the GEBCO Digital Atlas, GEBCO, 2003.

Greene, N.: Environment mapping and other applications of world projections, IEEE Comput. Graph., 6, 21–29, doi:10.1109/MCG.1986.276658, 1986.

Greiner, B.: Euler rotations in plate-tectonic reconstructions, Comput. Geosci., 25, 209–216, 1999.

Gurnis, M., Turner, M., Zahirovic, S., Dicaprio, L., Spasojevich, S., Muller, R. D., Boyden, J., Seton, M., Manea, V. C., and Bower, D. J.: Plate reconstructions with continuously closing plates, Comput. Geosci., 38, 35–42, doi:10.1016/j.cageo.2011.04.014, 2012.

Kumar, M.: World geodetic system 1984: a modern and accurate global reference frame, Mar. Geod., 12, 117–126, doi:10.1080/15210608809379580, 1988.

Landgrebe, T. C. W. and Muller, R. D.: A Spatio-Temporal Knowledge-Discovery Platform for Earth-Science Data, Digital Image Computing Techniques and Applications (DICTA), 2011 International Conference on 6–8 December 2011, 394–399, doi:10.1109/DICTA.2011.73, 2011.

Müller, R. D., Sdrolias, M., Gaina, C., and Roest, W. R.: Age, spreading rates and spreading asymmetry of the world's ocean crust, Geochem. Geophy. Geosy., 9, Q04006, doi:10.01029/02007GC001743, 2008.

Neider, J., Davis, T., and Woo, M.: OpenGL, Programming Guide, Addison-Wesley, 1997.

OpenGL Architecture Review Board: OpenGL Shading Language, available at: http://www.opengl.org/documentation/glsl/ (last access: 22 January 2014), 2014.

Qin, X., Müller, R, D., Cannon, J., Landgrebe, T. C. W., Heine, C., Watson, R. J., and Turner, M.: The GPlates geological information model and markup language, geoscientific instrumentation, Methods Data Systems, 1, 111–134, doi:10.5194/gi-1-111-2012, 2012.

Sagan, H.: Hilbert's Space-Filling Curve, in: Space-Filling Curves, Universitext, Springer, New York, 9–30, 1994.

Seton, M., Müller, R., Zahirovic, S., Gaina, C., Torsvik, T., Shephard, G., Talsma, A., Gurnis, M., Turner, M., and Maus, S.: Global continental and ocean basin reconstructions since 200 Ma, Earth-Sci. Rev., 113, 212–270, 2012.

Warmerdam, F.: The Geospatial Data Abstraction Library, in: Open Source Approaches in Spatial Data Handling, edited by: Hall, G. B. and Leahy, M., Advances in Geographic Information Science, Springer, Berlin, Heidelberg, 87–104, 2008.

Williams, L.: Pyramidal parametrics, ACM Siggraph Computer Graphics, 17, 1–11, 1983.

Williams, S. E., Muller, R. D., Landgrebe, T. C. W., and Whittaker, J. M.: An open-source software environment for visualizing and refining plate tectonic reconstructions using high-resolution geological and geophysical data sets, GSA Today, 22, 4–9, doi:10.1130/GSATG139A.1, 2012.