International Journal of Digital Earth, 2015
http://dx.doi.org/10.1080/17538947.2014.1002867

Taylor & Francis
Taylor & Francis Group

Real-time raster projection for web maps

Bernhard Jenny*, Bojan Šavrič and Johannes Liem

College of Earth, Ocean, and Atmospheric Sciences, Oregon State University, Corvallis, OR, USA

(*Received 9 September 2014; accepted 23 December 2014*)

The virtual globe is the default visualization for Digital Earth applications, but it can only show one half of the Earth. This article introduces user-adjustable, on-the-fly projection of georeferenced raster images for web mapping and web GIS applications. This technique allows users to center a map on arbitrary locations, while still seeing the entire Earth surface. Modern web mapping libraries can apply map projection transformations to vector data in the browser, but they do not currently support the projection of raster images at interactive frame rates. We use the cross-platform WebGL (Web Graphics Library) graphics pipeline for hardware-accelerated projection of raster data in web browsers. Two algorithmic techniques – inverse per-fragment projection, and combined forward per-triangle and inverse per-fragment projection – for georeferenced raster imagery are documented. The resulting raster maps combine the ease of use and flexibility of interactive virtual globes with the ability to show the entire Earth. The adjustable projection of raster imagery allows users to observe global phenomena that are difficult to visualize on virtual globes or traditional maps with static projections.

Keywords: map projection; raster projection; WebGL

Introduction

The default visualization tool for Digital Earth applications is the virtual globe, a very powerful mechanism for visualization (Craglia et al. 2012). It enables smooth transitions between locations and data integration across a wide range of scales, from global to local (Goodchild 1999). Virtual globes, such as Google Earth, avoid traditional cartographic projections by depicting the Earth as seen from space. This results in visualizations that can only depict half of the Earth, and additionally severely distort shape and area along the border of the displayed hemisphere (Snyder 1987). Plane world maps, in contrast, show the entire Earth surface and are the only means to visualize global phenomena with a synoptic view. We concur with Goodchild et al. (2012) that "there will always be a need for flattening the Earth, to see the entire surface at once," and we argue that the hemispheric orthographic projection of current virtual globes should be complemented with user-adjustable projections that show the entire Earth at one time. Users need to be able to center a map on arbitrary locations because maps centered on the equator are unable to effectively visualize all aspects of many global phenomena. An example is the visualization of global warming models: A global warming map should allow the user to move one of the poles toward the map center to better visualize essential circumpolar patterns. At the same time, this map should depict the whole Earth surface (preferably

*Corresponding author. Email: jennyb@geo.oregonstate.edu

without area distortion) to show the entirety of this global phenomenon in a comprehensive image. However, there is currently no technique available for web-based visualization of raster imagery that can center maps on arbitrary locations, and little research exists that focuses on the real-time projection of raster data for two-dimensional (2D) web maps. A variety of three-dimensional (3D) virtual globes with raster image texturing for web browsers have been introduced, and algorithms for the real-time projection and rendering of georeferenced vector data have been developed for web mapping (Bostock and Davies 2013). However, none of the available web mapping libraries currently support the real-time projection of georeferenced raster imagery.

This article introduces two techniques for real-time hardware-accelerated raster projection in web browsers. The user can adjust the projection or projection parameters, for example, to place one of the poles at the center of the map and still see the entire surface of the Earth in a single visualization. These techniques allow for new interpretative insights in the context of Digital Earth, as global scientific datasets can be visualized with various viewpoints (e.g., polar or oblique map aspects).

With the presented results, we also hope to make a contribution to the development of alternatives to the static web Mercator projection, which is a poor choice for maps at medium and global scales because it creates a highly distorted portrait of the world (Battersby et al. 2014). An alternative can be found in program logic that autonomously selects and configures an appropriate projection, such as the adaptive composite map projection technique (Jenny 2012). Therefore, possible applications of the techniques presented in this article for real-time projection of georeferenced imagery include a variety of interactive Digital Earth applications, as well as desktop and web-based GIS.

This article first discusses related previous work, including applications of WebGL (Web Graphics Library), a JavaScript API (application programming interface) for rendering interactive 3D and 2D graphics in web browsers. The following section proposes two approaches for projecting raster imagery. The first approach uses inverse projection; the second approach uses a combination of forward and inverse projections. The discussion section evaluates the proposed methods, focusing on techniques for eliminating aliasing artifacts and rendering speed. The final section briefly addresses possible future extensions.

Related work

Projecting vector data is possible with web mapping libraries although it is infrequently used. Web browsers can project and render vector map data in real time using the JavaScript programming language and a dedicated projection pipeline (Jenny and Šavrič 2013). D3 is an example of a library with geospatial vector data projection (Bostock, Ogievetsky, and Heer 2011; Bostock and Davies 2013), which uses different technologies for rendering the projected vector map data, for example, SVG (Scalable Vector Graphics) or HTML5 Canvas (World Wide Web Consortium 2011; WHATWG 2014; Gaffuri 2012; Lienert et al. 2012).

The projection of raster data and associated resampling and aliasing issues causing information loss and creating visual artifacts are a classical problem in GIScience (Steinwand, Hutchinson, and Snyder 1995). In addition, high-quality projection of raster data is computationally expensive and therefore generally slow. Finn et al. (2012b) describe the projection of raster data using inverse projection equations, a technique commonly used by geospatial software. It is important to distinguish raster imagery with

discrete categorical information and continuous tone color information. The projection of discrete raster datasets with categorical information requires optimized nearest neighbor sampling in order to preserve categorical values, which is required for subsequent categorical analysis (Mulcahy 2000; Usery and Seong 2001; Kimerling 2002; Steinwand 2003; Usery et al. 2003; Bauer-Marschallinger, Sabel, and Wagner 2014). Our research focuses on the projection of raster images with continuous tone colors, including, for example, shaded relief images, land cover data with smooth transitions, or precipitation raster data symbolized with a color ramp. Different interpolation methods can be used for resampling this type of imagery.

An attempt at accelerating raster projection has been presented by Xie et al. (2011). They compute exact analytical projection transformations for a subset of pixels, and apply polynomial interpolation to the remaining pixels using the first subset as control points. Their algorithm is designed for very large images and applications where geometric accuracy can be sacrificed for speed of execution. Tang and Feng (in press) describe the hardware-accelerated projection of big vector datasets using cloud computing and graphics processing units (GPUs). Their work is designed for off-line projection of vector data using general-purpose computing on GPUs. GIS and specialized projection software, such as Geocart (Mapthematics 2014), use CPU-based multi-threading to accelerate projection computations (Kelso 2010). Finn et al. (2012a) extend this approach and developed a parallel projection program designed to scale to a supercomputer with thousands of processing units (Guan et al. 2009; Behzad et al. 2012). Their software is optimized for raster datasets up to several gigabytes in size. Zhao et al. (2011) describe the projection of large raster images with CUDA (Compute Unified Device Architecture). CUDA is a proprietary programming model designed by NVIDIA – a manufacturer of GPUs – that gives programs access to the parallel architecture of GPUs. These related approaches provide inspiration for our work, but are not directly applicable to web maps. In web maps, datasets are often smaller, but the projection must be applied in real time at interactive frame rates for a responsive user interface.

Hardware acceleration is required to achieve interactive frame rates when projecting raster data. Hardware acceleration for graphics applications in web browsers is accessible through WebGL, a JavaScript API for rendering 2D and 3D graphics based on OpenGL ES 2.0, a subset of the widely used graphics library OpenGL (Khronos Group 2014). The WebGL standard was released in 2011 and is integrated in all major desktop and mobile web browsers. Resch, Wohlfahrt, and Wosniok (2014) compared WebGL to other technologies suitable for 3D web cartography. They compared Java 3D (accessible through Java applets), Silverlight (based on the Microsoft.Net framework), Stage 3D (for Adobe Flash/AIR), and the Unity Web Player game engine. They found that the major advantage of WebGL compared to the other technologies is that it runs natively in web browsers, is platform-independent, and does not require the installation of an additional browser plug-in. Resch, Wohlfahrt, and Wosniok (2014) also point out that some WebGL functionality shows different behavior on different browsers, which is due to bugs in browser and driver software.

WebGL provides access to the hardware-accelerated graphics pipeline of the GPU through a JavaScript API. The GPU is a collection of a large number of specialized processors that run in parallel and are therefore able to concurrently operate on a large number of geometry vertices. Raster images could also be projected with the considerably slower JavaScript language. However, JavaScript cannot run on a GPU, and raster

projection with JavaScript can therefore not be as effectively parallelized and hardware accelerated as with WebGL.

Various libraries based on WebGL have been introduced to abstract the complexity of the underlying technology and accelerate the development of applications (Levkowitz and Kelleher 2012). Some specialized libraries are available for developing geospatial interactive virtual globes with WebGL. Virtual globes developed with WebGL include Glob3 Mobile (Suárez et al. 2012), WebGL Earth (Klokan Technologies 2013), OpenWebGlobe (Loesch, Christen, and Nebiker 2012; Christen, Nebiker, and Loesch 2012), Nokia HERE (previously Nokia Maps for WebGL; Nokia 2013), WebGL Globe (Google Data Arts Team 2014), and Cesium (Analytical Graphics 2013). Cozzi and Ring (2011) describe techniques, challenges and pitfalls when developing virtual globes with a graphics pipeline, and Cozzi and Bagnell (2013) describe issues specific to the development of a WebGL rendering engine for virtual globes.

WebGL is also used for a variety of other WebGIS, cartography, and geovisualization applications. Examples include 3D WebGIS (Feng et al. 2011), campus information systems (Hering et al. 2011), relief shading (Auer 2012), visualization of 3D city models (Gesquière and Manin 2012), rendering of street views (Devaux, Brédif, and Paparoditis 2012), computation of raster images with inverse distance weighting (Lienert, Bär, and Hurni 2013), rendering 2D maps (OpenLayers 3 2014) and 3D maps (Stamen 2013), visualization of satellite images (Kim et al. 2014), or rendering terrain maps with user-adjustable plan oblique relief projection (Jenny et al. in press).

The OpenGL Shading Language is used to develop programs that are executed on the GPU. These programs are called shader programs, and they control the rendering process. In WebGL, two types of shader programs exist: vertex shaders, which can manipulate vertices of the geometry model; and fragment shaders, which determine the color of image fragments. (In computer graphics, a fragment is the data required for generating the color of a pixel.) Both types of shader programs can read color values from texture images, which are raster images loaded onto the GPU.

Methods

When developing raster projection for web browsers, we identified a set of requirements. The projection must be fast enough to enable interactive frame rates. For a smooth user experience when panning or zooming on a map, rates of 30 frames per second or more are desirable. We require high-quality raster resampling with a minimum of visual artifacts, such as aliasing or blurring. We also require the technique to be available on diverse operating systems and web browsers, as well as on diverse hardware, including mobile devices. Finally, diverse projections must be supported, including world map projections with oblique aspects and arbitrary central meridians, as well as common projections for medium scales.

Based on these requirements, we developed two methods for the projection of raster images with WebGL. The first technique is an inverse per-fragment projection method. This method uses a very simple geometry model consisting of a single flat rectangle covering the entire viewport (the viewing region of a window for rendering the map). The projection logic is placed in the fragment shader, which computes an individual color for each fragment. The second technique is a forward per-triangle projection of a sphere model combined with inverse per-fragment projection. Here, the geometry consists of a tessellated sphere. The projection logic is in the vertex and fragment shader programs.

Inverse per-fragment projection

The inverse per-fragment projection method is the technique usually applied to the projection of raster data (Finn et al. 2012b). For each fragment of the output image, we compute the corresponding coordinates on the sphere using inverse projection equations. Inverse projection equations convert from Cartesian X/Y coordinates to spherical longitude and latitude. The geometry model for WebGL consists of two triangles that cover the entire viewport. The vertex shader is simple; it only passes the unaltered vertices of the two triangles to the rasterization process. The projection logic is placed in the fragment shader program, which executes the following transformation steps for each fragment:

1. The window-relative coordinates of the fragment are divided by the nominal map scale to become relative to the coordinate space of the projected generating globe with a radius of 1.
2. False northing and false easting values are subtracted from the Cartesian coordinates.
3. The projection-specific inverse equations are applied to convert the Cartesian X/Y coordinates to spherical λ/φ coordinates. If the location is outside of the valid range defined by the projection, no color is assigned to the fragment, and the following steps are not executed.
4. The spherical λ/φ coordinates are rotated in the inverse direction on the sphere if an oblique aspect is created (for equations, see Snyder 1987, 29ff.).
5. The longitude of the central meridian is added to the longitude λ.
6. The spherical λ/φ coordinates are converted to texture coordinates (commonly called UV texture mapping with U and V denoting the axes of the 2D texture between 0 and 1) by scaling and shifting.
7. The UV texture coordinates are used to sample the fragment color from a texture image with Plate Carrée projection, using standard OpenGL functionality.

The false easting and false northing in Step 2 do not need to be subtracted for interactive maps when a drag event is not converted to a translation of the map, but instead converted to a spherical rotation. This is the case for maps where the user can adjust the central meridian and create oblique projection aspects.

Figure 1 shows an example of a raster image transformed with this procedure. Aliasing artifacts can become visible along the border of the map (Figure 2). The artifacts are due to the lack of alpha blending along the border of the map. Alpha blending is applied by the graphics pipeline along visible triangle borders when rasterizing the triangles, and has the effect of slightly blending colors to avoid pixelated aliasing. However, the geometry for inverse per-fragment projection consists of two triangles covering the entire viewport. Hence, alpha blending is not applied along the border of the graticule. Instead, the fragment shader samples color values from the texture for fragments inside the graticule and assigns a background color to fragments outside the graticule. This creates an abrupt transition between the inner and outer graticule area. To avoid the resulting pixelated border, the fragments would have to be subdivided along the graticule border, and the relative area of each fragment inside the border would have to be computed. This would be required for all fragments within a small one-pixel wide area along the border of the graticule. The relative area could then be used for alpha blending color values along the border. This is possible for virtual globes, which have a circular

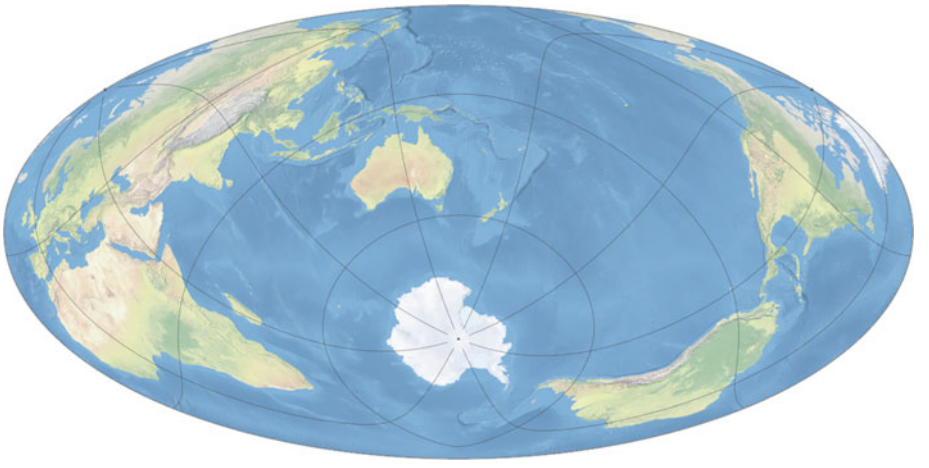

Figure 1. Inverse per-fragment transformation with WebGL; graticule drawn with HTML5 Canvas. Oblique Hammer projection.

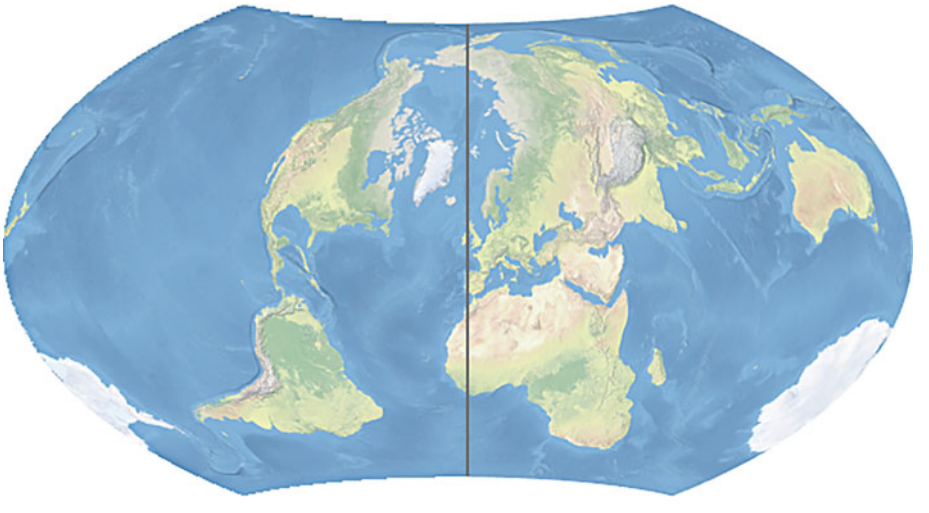

Figure 2. Inverse per-fragment projection creates pixelated artifacts along map borders (left); alpha blending with the combined method results in anti-aliased map borders (right). Oblique Wagner VII projection.

outline, but is considerably more complicated for many projections that have a more irregular graticule outline. In addition, alpha blending values need to be computed for fragments that are close to the graticule border but outside the range of ±180° longitude and ±90° latitude. However, many projection equations are undefined for positions outside this range. Two alternative approaches are possible to create anti-aliased map borders. (1) The geometry consisting of two triangles forming a rectangle can be replaced by a more complex shape following the outline of the graticule. The graphics pipeline then alpha-blends the border of this shape with the background color. (2) The alpha

channel of the aliased map can be blended along the border with a moving window filter, for example, with a 3 × 3 blur filter. Both techniques require additional computations that inevitably have an impact on interactive frame rates. Alternatively, the aliased border can be covered by a vector outline drawn along the border of the graticule. This solution is by far the simplest approach and is applied in Figure 1.

Inverse projection equations are available for many projections, though a few lack direct inverse equations, such as the Robinson or the Winkel Tripel projections. For these projections, the inverse transformation can be computed with iterative methods, such as the Newton-Raphson root finding method (Ipbuker and Bildirici 2002; Ipbuker 2009). These iterative equations are substantially slower to execute than the forward equations. For some projections, however, the inverse equations are also substantially simpler and faster to execute than the forward equations. This is the case, for example, for the equal-area Mollweide and Eckert IV projections, both of which require iterative computations for the forward equations but have direct inverse equations (Snyder 1987).

Combined forward per-triangle and inverse per-fragment projection

An alternative to the inverse per-fragment projection method described above uses a globe model that is forward-projected into a plane. This approach uses a tessellated vector sphere model. Cozzi and Ring (2011) discuss various tessellation schemas for rendering virtual globes, such as subdivision surfaces, cube-map tessellation, and geographic-grid tessellation, which can also be applied to forward per-triangle map projection. They point out that geographic-grid tessellation, that is, tessellation along parallels and meridians, creates overcrowding of triangles near poles. The thin triangles near poles can cause lighting and texturing artifacts and performance issues due to oversampling. However, the geographic-grid tessellation is not necessarily a limitation for rendering maps. Many projections represent the poles with pole lines, and triangles are therefore not overly thin with such projections. For projections that show poles as points, the resolution of the tessellation can be adjusted with latitude (Gerstner 2003). Cozzi and Ring (2011) also discuss algorithmic details for the efficient creation of triangle strips to optimize rendering performance for tessellated spheres.

A simplistic approach to creating a flat map from a spherical geometry model would consist of the following three steps: (1) Subtracting the longitude of the central meridian from each vertex. This corresponds to a rotation around the Earth's axis that brings the central meridian to the center of the map. (2) Rotating the spherical λ/φ coordinates to create an oblique aspect. (3) Forward-projecting the spherical coordinates to Cartesian coordinates. The texture coordinates would be computed by linearly mapping the original (un-rotated) spherical coordinates to UV coordinates. This simplistic approach suffers from a major shortcoming – rotating the geometry model by the central meridian results in triangles that do not align with the border of the graticule, resulting in visual artifacts along the map border (Figure 3, left). These artifacts can be reduced by decreasing the size of the triangles in the tessellated sphere. However, reducing the triangle size cannot avoid all visual artifacts. For example, cylindrical projections and projections with pronounced edgy corners or curved pole lines would display rendering artifacts because the triangle vertices are unlikely to perfectly align with the outline of the map (Figure 3, left).

To avoid the described limitations, we propose an alternative approach, which does not rotate spherical coordinates before the projection equations are applied. Instead, each

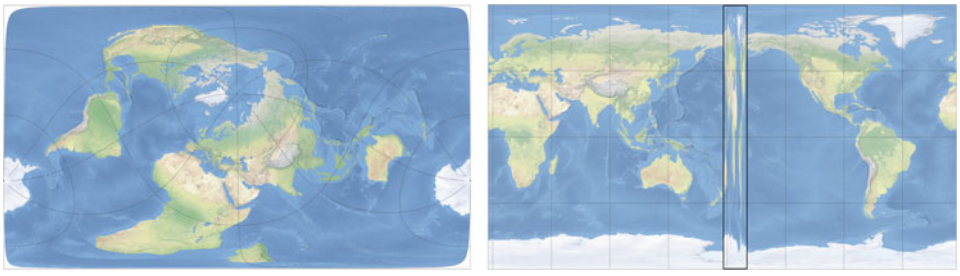

Figure 3. Left: Rotating and forward-projecting sphere coordinates creates artifacts along graticule borders, showing as white corners with an oblique Plate Carrée projection. Right: Rotating texture coordinates results in the entire texture image being mapped to a triangle strip along the anti-meridian at ±180°. Oversized triangles with a side length of 20° are used to highlight the problem.

geometry vertex is transformed from spherical λ/φ coordinates to Cartesian coordinates using the forward equatorial-aspect projection equation. When computing *UV* texture coordinates, the rotation and central meridian shifting are applied to the spherical coordinates – but the projection equations are not applied. This approach creates a projected raster image that aligns with the border of the graticule. However, a visual artifact appears along the anti-meridian at ±180°, where almost the entire texture image is mapped onto a narrow strip of triangles (Figure 3, right). This is a well-known problem when rendering virtual globes (Cozzi and Ring 2011). It can be solved for globes by repeated texture wrapping and modifying a horizontal texture coordinate of the triangle vertex that crosses the entire texture. For map rendering, this problem can be solved by rendering the triangles along the anti-meridian using the inverse per-fragment projection method, as described in the previous section. Hence, the vertex shader program executes the following transformation steps on each sphere vertex (which is initially in spherical λ/φ coordinates):

1. Spherical λ/φ coordinates are projected to Cartesian coordinates, and the map scale, false northing, and false easting are applied. The resulting Cartesian coordinates correspond to the vertex position on the map.
2. The original spherical λ/φ coordinates are rotated on the sphere if an oblique aspect is created.
3. The longitude of the central meridian is added to the longitude λ.
4. The rotated spherical coordinates λ/φ are converted to *UV* texture coordinates (between 0 and 1) by scaling and shifting. The *UV* coordinates are passed to the fragment shader.
5. A flag is set to 1 if the current vertex is part of a triangle along the anti-meridian. Otherwise the flag is set to 0. This value is passed to the fragment shader.

The fragment shader first tests the flag value that may have been set to 1 by the fragment shader in Step 5. If the value is 0, the passed *UV* texture coordinates are used to sample the fragment color from the texture image. If the value is greater than 0, the fragment is within a triangle along the anti-meridian, and the fragment shader executes the transformation steps for inverse per-fragment raster projection described in the previous section. (The flag values are between 0 and 1 because the OpenGL rendering pipeline interpolates the flag values inside each triangle before calling the fragment shader.) The result is a projected raster image without visual artifacts as shown in Figure 4.

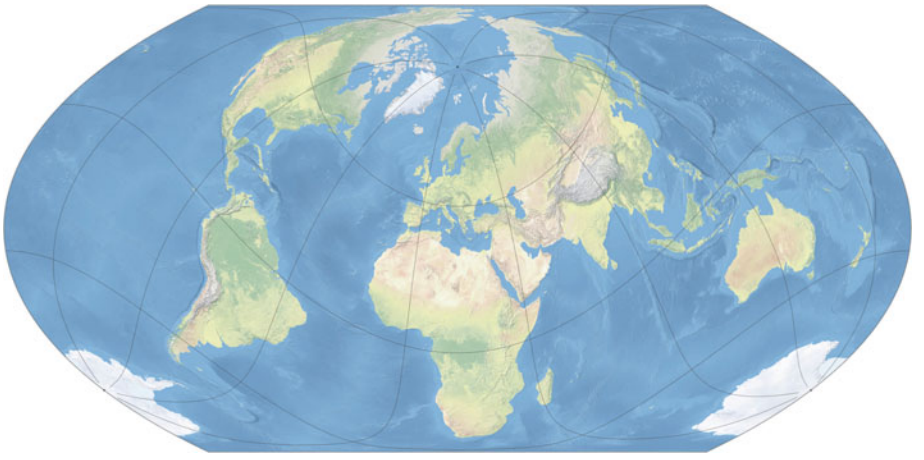

Figure 4. World map created with combined forward per-triangle and inverse per-fragment projection. Oblique equal-area pseudocylindrical projection after Šavrič and Jenny (2014).

If this method is used when only a section of the globe is visible, a potentially large number of projected sphere vertices are located outside the viewport. This causes two problems: First, computational resources are wasted by projecting points that are never visible; and second, the projected triangles inside the viewport are relatively large, which introduces spatial distortion due to the linear texture mapping applied inside each triangle. The two issues can be avoided by adjusting the extent of the spherical geometry to the geographic extent displayed in the viewport. To do so, the vertex shader computes the spherical coordinates λ/φ from a generic 2D grid that is used instead of the spherical geometry model. By scaling and shifting the vertices, the grid is placed such that it covers an area on the sphere that corresponds to the area displayed in the viewport. This technique ensures that most projected triangles are displayed on the map, and that distortion due to texture mapping inside the triangles is minimized.

Discussion

Anti-aliasing with mipmaps and anisotropic filtering

A square-shaped pixel in the projected map image corresponds to a non-square area that overlaps one or more pixels in the source image. In the general case, the square pixel corresponds to a shape consisting of four concave or convex lines in the source image. For sampling an appropriate color from the source image, multiple neighboring pixels have to be transformed (e.g., bilinear interpolation requires four neighboring pixels), and the overlapping areas between the projected irregularly shaped areas with the square-shaped source pixels would have to be computed for exact results. These calculations would result in fairly complex and computationally expensive algorithms. For both the inverse per-fragment and the combined forward-projection methods, we instead use mipmapping (Williams 1983) and anisotropic filtering, two texture mapping techniques for reducing aliasing effects that are integrated in graphics pipelines. The effect of the two techniques for raster projection is illustrated in Figure 5, where a relatively large image in Plate Carrée projection is used to texture a relatively small map. In Figure 5 (left), bilinear resampling is used, which results in excessive aliasing artifacts. Figure 5 (center) uses

Figure 5. Texture sampling: noisy bilinear sampling (left), trilinear mipmap sampling, blurred around poles (center), and trilinear mipmap sampling with anisotropic filtering (right). Screen captures are enlarged for visualizing graphical artifacts.

resampling based on a mipmap. A mipmap is an image pyramid, consisting of downsized images, with each image measuring half the size of its immediate, larger neighbor. Image pyramids are widely used in GIS – mipmap pyramids are the GPU-accelerated equivalent in computer graphics. When extracting a color from the pyramid, the graphics pipeline applies trilinear interpolation, which avoids aliasing artifacts, but adds blurring. Blurring occurs in areas where a square pixel of the source texture is projected to an elongated shape in the projected destination map. For maps with geographic-grid tessellation (Cozzi and Ring 2011), this is the case for polar regions (Figure 5, center). The square pixels in the Plate Carrée projection near the poles are massively compressed along the longitudinal axis when transformed to the oblique projection in Figure 5. The resulting blurring can be eliminated by anisotropic filtering, a technique developed for enhancing the quality of textures on surfaces viewed under oblique angles in 3D scenes. Anisotropic filtering samples the texture at more than one location when needed. The additional sampling may slightly slow down rendering performance, but increases image quality. The details of both anisotropic filtering (Nvidia 1999) and mipmapping in OpenGL are implementation-dependent and not documented in detail by GPU vendors. Figures 1 through 4, and 6, use mipmapping and anisotropic filtering.

The current version of WebGL does not support double precision floating-point numbers; generally only single floating-point precision is supported. This could be a potential limitation for some projections; however, we have not encountered stability or graphical issues with commonly used map projections, such as Hammer, Sinusoidal, Wagner VII, Robinson, Lambert azimuthal, Albers conic, or Mercator. With these projections, no visual artifacts appeared, and the raster data visually aligned with vector data projected with JavaScript double precision arithmetic and rendered with HTML5 Canvas. Precision artifacts might appear with other projections, at larger scales or with inverse per-fragment projection on mobile GPUs, which sometimes have less than single-precision floating-point arithmetic available to their fragment shaders (Ring 2013).

Rendering speed

The inverse per-fragment projection method is algorithmically simpler, but it can be computationally more expensive because each fragment requires an inverse projection. With the combined projection method, a projected texture coordinate is computed for each vertex in the triangle mesh by the shader program. The projected texture coordinates are interpolated by the graphics pipeline inside each triangle, which is computationally cheaper. Hence, computational expenses can be reduced by increasing the size of the

triangles in the geometry model. However, if the triangles chosen are too large, OpenGL's linear texture mapping inside the triangles causes spatial distortion (which is an extra type of distortion not related to distortion caused by the map projection). The amount of triangular distortion varies with the size of the projected triangle, which, in turn, varies with the map projection and the location on the graticule. However, adjusting the number of triangles allows for adapting to the available computing capacities. On a mobile device with a less powerful GPU, the number of triangles can be reduced for a smooth user experience, when, for example, the user adjusts the central meridian of the map and the map updates in real time. Once the central meridian is selected and the map is no longer animated, smaller triangles can be rendered for creating a static image with less geometric distortion.

We achieve interactive frame rates on all tested devices. For example, for maps with a size of 2500 × 1300 pixels on screen, the number of frames drawn per second is between 40 and 60 with both the per-fragment inverse projection method, as well as the combined forward and inverse projection method (with an NVIDIA GeForce GT 650M GPU). The measurements include the time required for four consecutive operations: (1) the JavaScript event handler code is called, (2) the projection parameters for the aspect adaptive projection framework are computed (Jenny 2012), (3) parameters are passed to the vertex and fragment shader programs, and (4) raster images are projected with the shader programs. In comparison, the projection of raster images with JavaScript is considerably slower. For example, a single-threaded JavaScript version of the same projection applied to a raster image for a map of the same size required approximately 0.75 seconds with a 2.3 GHz Intel Core i7 CPU and various modern browsers. Aliasing artifacts were visible in the JavaScript version, because we used a computationally simple nearest-neighbor sampling technique without mipmapping and anisotropic filtering.

The interactive frame rates reported above allow for smoothly animated maps that follow user interaction. This is relevant in the context of the aspect adaptive projection framework (Jenny 2012), which selects projections and projection parameters to minimize distortion for the area shown in the map. Whenever the map user pans or zooms the map, the projection parameters (such as the central meridian or standard parallels) are adjusted, and raster and vector data are re-projected. This results in an animated map projection that smoothly adapts to user adjustments. The four steps outlined above (event handler dispatch, projection parameterization, shader program configuration, and raster projection) are executed for each frame of this animation while the user adjusts the scale or central location of the map.

Conclusion

The real-time projection of raster data offers new opportunities for web mapping, WebGIS, and desktop GIS applications. It combines the ease of use and flexibility of interactive virtual globes with the ability to show the entire Earth. The WebGL technology used in the presented implementation allows for high-quality, artifact-free visualization of raster images. With the combination of the forward per-triangle and inverse per-fragment projection method, no visual artifacts can be detected, and aliasing along the border of the graticule can be avoided. Furthermore, very diverse projections for small and large map scales can be implemented with both raster projection methods. Hence, all of the identified requirements for web maps are met. The presented techniques

are also applicable to desktop GIS applications and other computing platforms with different graphics pipelines, such as MS Windows Direct3D.

The presented methods use raster source images in the Plate Carrée projection. Alternatively, additional source projections could be supported, such as the Sinusoidal map projection (which is recommended for global raster data by Mulcahy [2000], Usery and Seong [2001], and others), equidistant projections (which are recommended for sampling gridded data by Bauer-Marschallinger, Sabel, and Wagner [2014]) or the web Mercator projection (which is the standard used by tiled web maps). Another area of future research is the real-time projection of discrete categorical raster data, which requires optimized nearest neighbor sampling in order to preserve categorical values.

The techniques presented in this article extend the adaptive composite map projections, which were proposed as an alternative to the Mercator projection for web maps, but have only supported the real-time projection of vector data so far (Jenny 2012; Šavrič and Jenny 2014). Adaptive composite map projections could also be embedded in desktop GIS software. Such software would not require users to choose a map projection – a process that is obscure and confusing to many GIS users – but instead automatically selects a projection that is optimized for the geographic area shown by the map.

We hope that developers of web mapping libraries will include WebGL-based projection of raster data in their frameworks to simplify the process of creating web maps with non-Mercator projections.

Acknowledgments

The support of Esri is greatly acknowledged, including valuable discussions with David Burrows, Scott Morehouse, and Dawn Wright. The authors would like to thank Jane E. Darbyshire and Abby Phillips Metzger, both from Oregon State University, for editing this article.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Analytical Graphics. 2013. "Cesium – WebGL Virtual Globe and Map Engine." Accessed July 25 2014. http://cesiumjs.org.

Auer, M. 2012. "Real-time web GIS Analysis using WebGL." *International Journal of 3-D Information Modeling* 1 (3): 49–61.

Battersby, S. E., M. P. Finn, E. L. Usery, and K. H. Yamamoto. 2014. "Implications of Web Mercator and its Use in Online Mapping." *Cartographica: The International Journal for Geographic Information and Geovisualization* 49 (2): 85–101. doi:10.3138/carto.49.2.2313.

Bauer-Marschallinger, B., D. Sabel, and W. Wagner. 2014. "Optimisation of Global Grids for High-resolution Remote Sensing Data." *Computers & Geosciences* 72: 84–93. doi:10.1016/j.cageo.2014.07.005.

Behzad, B., Y. Liu, E. Shook, M. P. Finn, D. M. Mattli, and S. Wang. 2012. "A Performance Profiling Strategy for High-performance Map Re-projection of Coarse-scale Spatial Raster Data." Proceedings of AutoCarto 2012, 16–18 September 2012, Columbus, Ohio.

Bostock, M., and J. Davies. 2013. "Code as Cartography." *The Cartographic Journal* 50 (2): 129–135. doi:10.1179/0008704113Z.00000000078.

Bostock, M., V. Ogievetsky, and J. Heer. 2011. "D³ Data-driven Documents." *Visualization and Computer Graphics, IEEE Transactions on* 17: 2301–2309. doi:10.1109/TVCG.2011.185.

Christen, M., S. Nebiker, and B. Loesch. 2012. "Web-based Large-scale 3D-Geovisualisation Using WebGL: The OpenWebGlobe Project." *International Journal of 3-D Information Modeling* 1 (3): 16–25.

Cozzi, P., and D. Bagnell. 2013. "A WebGL Globe Rendering Pipeline." *GPU Pro 4: Advanced Rendering Techniques* 4: 39–48.
Cozzi, P., and K. Ring. 2011. *3D Engine Design for Virtual Globes*. Boca Raton, FL: CRC Press.
Craglia, M., K. de Bie, D. Jackson, M. Pesaresi, G. Remetey-Fülöpp, C. Wang, A. Annoni, et al. 2012. "Digital Earth 2020: Towards the Vision for the Next Decade." *International Journal of Digital Earth* 5 (1): 4–21. doi:10.1080/17538947.2011.638500.
Devaux, A., M. Brédif, and N. Paparoditis. 2012. "A Web-based 3D Mapping Application using WebGL Allowing Interaction with Images, Point Clouds and Models." Proceedings of the 20th International Conference on Advances in Geographic Information Systems, 586–588, ACM.
Feng, L., C. Wang, C. Li, and Z. Li. 2011. "A Research for 3D WebGIS Based on WebGL." Proceedings of the International Conference on Computer Science and Network Technology, Harbin, December 24–26, 348–351.
Finn, M. P., Y. Liu, D. M. Mattli, Q. Guan, K. H. Yamamoto, E. Shook, and B. Behzad. 2012a. "pRasterBlaster: High-performance Small-scale Raster Map Projection Transformation Using the Extreme Science and Engineering Discovery Environment." Abstract presented at the XXII International Society for Photogrammetry & Remote Sensing Congress, Melbourne, Australia.
Finn, M. P., D. R. Steinwand, J. R. Trent, R. A. Buehler, D. M. Mattli, and K. H. Yamamoto. 2012b. "A Program for Handling Map Projections of Small Scale Geospatial Raster Data." *Cartographic Perspectives* 71 (71): 53–67. doi:10.14714/CP71.61.
Gaffuri, J. 2012. "Toward Web Mapping with Vector Data." In *Geographic Information Science*, edited by X. Ningchuan, M.-P. Kwan, M. F. Goodchild, and S. Shekhar, 87–101. Berlin: Springer.
Gerstner, T. 2003. "Multiresolution Visualization and Compression of Global Topographic Data." *GeoInformatica* 7 (1): 7–32. doi:10.1023/A:1022818126783.
Gesquière, G., and A. Manin. 2012. "3D Visualization of Urban Data based on CityGML with WebGL." *International Journal of 3-D Information Modeling* 1 (3): 1–15.
Goodchild, M. F. 1999. "Implementing Digital Earth: A Research Agenda." Towards Digital Earth: Proceedings of the International Symposium on Digital Earth (Vol. 29). 29 November–2 December, Beijing.
Goodchild, M. F., H. Guo, A. Annoni, L. Bian, K. de Bie, F. Campbell, M. Craglia, et al. 2012. "Next-generation Digital Earth." *Proceedings of the National Academy of Sciences* 109: 11088–11094. doi:10.1073/pnas.1202383109.
Google Data Arts Team. 2014. "The WebGL Globe." www.chromeexperiments.com/globe.
Guan, Q., M. P. Finn, E. L. Usery, and D. M. Mattli. 2009. "Rapid Raster Projection Transformation and Web Service using High-performance Computing Technology (Abstract)." Presented at the Association of American Geographers Annual Meeting, Las Vegas, NV.
Hering, N., M. Rünz, L. Sarnecki, and L. Priese. 2011. "3DCIS: A Real-time Browser-rendered 3D Campus Information System based on WebGL." The 2011 World Congress in Computer Science, Computer Engineering and Applied Computing. Worldcomp, 18–21.
Ipbuker, C. 2009. "Inverse Transformation for Several Pseudo-cylindrical Map Projections Using Jacobian Matrix." In *Computational Science and its Applications–ICCSA 2009*, edited by O. Gervasi, D. Taniar, B. Murgante, A. Laganà, and Y. Mun, 553–564. Berlin: Springer.
Ipbuker, C., and I. O. Bildirici. 2002. "A General Algorithm for the Inverse Transformation of Map Projections using Jacobian Matrices." Proceedings of the Third International Conference on Mathematical & Computational Applications, Konya, Turkey, September 4–6, 175–182.
Jenny, B. 2012. "Adaptive Composite Map Projections." *Visualization and Computer Graphics, IEEE Transactions on* 18: 2575–2582.
Jenny, B., J. Buddeberg, C. Hoarau, and J. Liem. (in press). "Plan Oblique Relief for Web Maps." *Cartography and Geographic Information Science*.
Jenny, B., and B. Šavrič. 2013. "Rendering Vector Geometry with Adaptive Composite Map Projections." Proceedings of the 26th International Cartographic Conference, Dresden, Germany, 25–30 August 2013.
Kelso, N. V. 2010. "Review: Geocart 3." http://kelsocartography.com/blog/?p=3779.
Khronos Group. 2014. "WebGL – OpenGL ES 2.0 for the Web." Accessed July 25, 2014. http://www.khronos.org/webgl.

Kim, H. W., D. S. Kim, Y. W. Lee, and J. S. Ahn. 2014. "3-D Geovisualization of Satellite Images on Smart Devices by the Integration of Spatial DBMS, RESTful API and WebGL." *Geocarto International* (ahead-of-print): 1–19.

Kimerling, A. J. 2002. "Predicting Data Loss and Duplication When Resampling from Equal-angle Grids." *Cartography and Geographic Information Science* 29 (2): 111–126. doi:10.1559/152304002782053297.

Klokan Technologies. 2013. "WebGL Earth – Open Source 3D Digital Globe Written in JavaScript." Accessed July 25, 2014. http://www.webglearth.org.

Levkowitz, H., and C. Kelleher. 2012. "Cloud and Mobile Web-based Graphics and Visualization." Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on 21–35, IEEE.

Lienert, C., H. R. Bär, and L. Hurni. 2013. "GPU-accelerated Spatial Interpolation Rendering for Web-based Environmental Monitoring." Proceedings of the 26th International Cartographic Conference, Dresden, Germany, 25–30 August 2013.

Lienert, C., B. Jenny, O. Schnabel, and L. Hurni.. 2012. "Current Trends in Vector-based Internet Mapping: A Technical Review." In *Online Maps with APIs and WebServices, Lecture Notes in Geoinformation and Cartography*, edited by M.P. Peterson, 23–36. Berlin Heidelberg: Springer-Verlag.

Loesch, B., M. Christen, and S. Nebiker. 2012. "OpenWebGlobe–An Open Source SDK for Creating Large-scale Virtual Globes on a WebGL Basis." *International Archives of the Photogrammetry Remote Sensing and Spatial Information Sciences XXII ISPRS Congress*.

Mapthematics. 2014. "Mapthematics Geocart 3." http://www.mapthematics.com.

Mulcahy, K. A. 2000. "Two New Metrics for Evaluating Pixel-based Change in Data Sets of Global Extent due to Projection Transformation." *Cartographica: The International Journal for Geographic Information and Geovisualization* 37 (2): 1–12. doi:10.3138/C157-258R-2202-5835.

Nokia. 2013. "here.com." http://here.com/3d.

Nvidia. 1999. "EXT_texture_filter_anisotropic." https://www.opengl.org/registry/specs/EXT/texture_filter_anisotropic.txt.

OpenLayers 3. 2014. "OpenLayers 3: A High-performance, Feature-packed Library for all your Mapping Needs." http://ol3js.org.

Resch, B., R. Wohlfahrt, and C. Wosniok. 2014. "Web-based 4D Visualization of Marine Geo-data using WebGL." *Cartography and Geographic Information Science* 41: 235–247. doi:10.1080/15230406.2014.901901.

Ring, K. 2013. "World-scale Terrain Rendering." Presentation for the Rendering Massive Virtual Worlds course, SIGGRAPH 2013. http://cesiumjs.org/massiveworlds/downloads/Ring/WorldScaleTerrainRendering.pptx.

Šavrič, B., and B. Jenny. 2014. "A New Pseudocylindrical Equal-area Projection for Adaptive Composite Map Projections." *International Journal of Geographical Information Science* 28 (12): 2373–2389. doi:10.1080/13658816.2014.924628.

Snyder, J. P. 1987. *Map Projections–A Working Manual* (No. 1395). USGPO.

Stamen. 2013. "Stamen <3 here: An Experimental Service using WebGL and 3D Data from here.com." http://here.stamen.com.

Steinwand, D. R., J. A. Hutchinson, and J. P. Snyder. 1995. "Map Projection for Global and Continental Data Sets and an Analysis of Pixel Distortion caused by Reprojection." *Photogrammetric Engineering & Remote Sensing* 61: 1487–1497.

Steinwand, D. R. 2003. "A New Approach to Categorical Resampling." Proceedings of the American Congress on Surveying and Mapping Spring Conference, Phoenix, AZ.

Suárez, J. P., A. Trujillo, M. de la Calle, D. D. Gómez-Deck, and J. M. Santana. 2012. "An Open Source Virtual Globe Framework for iOS, Android and WebGL Compliant Browser." *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications*, p. 22. ACM.

Tang, W., and W. Feng. in press. "Parallel Map Projection of Vector-based Big Spatial Data: Coupling Cloud Computing with Graphics Processing Units." *Computers, Environment and Urban Systems*. doi:10.1016/j.compenvurbsys.2014.01.001.

Usery, E. L., M. P. Finn, J. D. Cox, T. Beard, S., Ruhl, and M. Bearden. 2003. "Projecting Global Datasets to Achieve Equal Areas." *Cartography and Geographic Information Science* 30 (1): 69–79. doi:10.1559/152304003100010956.

Usery, L. E., and J. C. Seong. 2001. "All Equal-area Map Projections are Created Equal, but Some are More Equal than Others." *Cartography and Geographic Information Science* 28 (3): 183–194. doi:10.1559/152304001782153053.

WHATWG. 2014. "HTML Living Standard (4.12.4 The Canvas Element)." https://html.spec.whatwg.org/multipage/scripting.html#the-canvas-element.

Williams, L. 1983. "Pyramidal Parametrics." *ACM Siggraph Computer Graphics* 17 (3): 1–11. doi:10.1145/964967.801126.

World Wide Web Consortium. 2011. *Scalable Vector Graphics (SVG) 1.1*. 2nd ed. http://www.w3.org/TR/SVG/

Xie, Y., X. Tang, M. Sun, and H. Chen. 2011. "Multi-rank Method – Achieving Projection Transformation with Adjustable Accuracy and Speed." Proceedings of ISPRS Hannover Workshop 2009 High-Resolution Earth Imaging for Geospatial Information, Hannover, Germany.

Zhao, Y., Z. Cheng, H. Dong, J. Fang, and L. Li. 2011. "Fast Map Projection on CUDA." *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 4066–4069, IEEE.