MapCruncher: Integrating the World's Geographic Information

Jeremy Elson, Jon Howell, and John R. Douceur
Microsoft Research Redmond

jelson@microsoft.com

ABSTRACT

Current large-scale interactive web mapping services such as Virtual Earth and Google Maps use large distributed systems for delivering data. However, creation and editorial control of their content is still largely centralized. The Composable Virtual Earth project's goal is to allow seamless interoperability of geographic data from arbitrary, distributed sources.

MapCruncher is a first step in this direction. It lets users easily create new interactive map data that can be layered on top of existing imagery such as road maps and aerial photography. MapCruncher geographically registers and reprojects the user's map into a standard coordinate system. It then emits metadata that makes it easy for anyone on the Internet to find the published map data and import it. Interactive maps them become distributed, seamlessly composable building blocks – similar to images in the early days of the Web.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications – s*patial databases and GIS*; D.2.6 [**Software Engineering**]: Programming Environments – *graphical environments*; D.2.12 [**Software Engineering**]: Interoperability; G.4 [**Mathematical Software**].

Keywords

Interactive maps, composition, mashups, geographic coordinate systems, graphical interactive georeferencing, map projections, approximate reprojection, decentralized publishing, image tiling.

1. INTRODUCTION

In the relatively short time since the introduction of online mapping sites like Google Maps [8] and Microsoft Virtual Earth [16], hundreds of user-created "mashups" have appeared. These mashups cover a wide diversity of subjects. For example, Seattle Bus Monster [19] plots public transportation routes in Seattle; chicagocrime.org [2] highlights dangerous areas of Chicago; RunwayFinder [18] summarizes weather and airspace surrounding general aviation airports; housingmaps.com [11] shows real estate prices. These specialized sites each display the data from their particular application domain on top of maps and aerial imagery supplied by Google or Microsoft.

While useful individually, mashups can be far more useful when *integrated with each other*. Today, however, mashups are largely independent. For example, to find cheap real estate in a low-crime neighborhood, or find the public transportation near a general-aviation airport, users must visit each mashup individually and manually integrate the results. The goal of Microsoft Research's Composable Virtual Earth (CVE) project is to find new ways of constructing geographic Web mashups so that they can seamlessly interoperate.

Existing mashups are implemented largely in imperative code – JavaScript that runs in the client. This design gives site designers enormous flexibility, which led to the explosion of creative and innovative mashups. Well-known standards for describing geographically-tagged points, lines, and raster graphics had already existed for many years (e.g., GML [17], GeoRSS [6]); however, the sudden appearance of the mashups suggests many applications are not well-served by these standards. The combination of HTML and JavaScript allowed developers to go beyond creating layers, to create *applications*. In other words, mashup developers are using imperative code to customize exactly how their application operates, rather than simply creating layers declaratively whose user interactivity would be limited to "on" and "off."

A key design goal of CVE is to offer a mashup framework that is sufficiently structured to enable composition, yet sufficiently flexible to admit innovation. This interoperability balancing act is common in distributed systems design, from domain-specific frameworks such as the Flux OSKit [4], x-Kernel [15], and stackable file systems [9], to application-agnostic schemes such as Placeless active properties [3]. We plan to exploit the geographical domain constraints to best achieve this balance.

1.1 MapCruncher

As a first step, we created MapCruncher, a tool that allows users to add custom raster overlays onto the existing road and aerial imagery provided by Virtual Earth or Google Maps. Overlays are typically detailed maps, such as a bicycle route map, building floor plan, or campus map. The resulting web site is an interactive web map that features both the user's maps and the standard imagery. Like the underlying maps, user maps are pre-rendered into small image tiles at a variety of zoom levels, allowing the client to efficiently request the portions of a large virtual image that are needed for display.

MapCruncher first assists in registering the foreign map into the same (Mercator) coordinate system used by existing online map sites. Users select correspondence points between their own maps and existing maps, using road intersections or other recognizable landmarks. Once enough points are selected, MapCruncher estimates the transformation from the original map's coordinate system into Mercator by finding the best fit coefficients of a second-degree polynomial; while inexact, the error is typically small enough not to affect the results. MapCruncher then reprojects the original map and renders correctly registered and zoomed image tiles that can be seamlessly integrated with existing imagery.

Mashups created with MapCruncher do not restrict the developer's freedom to write arbitrary JavaScript that customizes the experience of their end-users, satisfying one of our design constraints for CVE. However, MapCruncher also emits metadata *about* the mashup, such as its geographic bounds, the file naming scheme for the tiles, and a brief description of the data as entered by the user. Because this data is semantically meaningful, it facilitates later discovery and integration of the imagery into other applications. In addition, much of this data is encoded as specially constructed strings that enable ordinary web search engines to find mashups matching geographic criteria. This combination of composability and discoverability takes us a step closer to our goal of a system that is capable of more seamless integration of geographic data on the web.

In the next section, we briefly describe the history of geographic mashups on the web. In Section 3, we review some of the difficulties in creating mashups using raster overlays. Section 4 describes approximate reprojection, the central technique used by MapCruncher to simplify the creation of raster mashups. In Section 5, we describe how this idea can be used to efficiently generate a database of image tiles. We review deployment issues and briefly describe a few sample applications in Section 6. Finally, we conclude in Section 7 with some thoughts on how MapCruncher might lead us towards an integrated and composable Virtual Earth.

2. INTERACTIVE WEB MAPS AND THE RISE OF MASHUPS

Online mapping services have existed for years. Until recently, they all had the same general architecture: maps would be custom-rendered from the underlying data on demand, in response to users' viewing requests. The only way to move the viewport was by clicking discrete buttons (e.g., "north"). The web service would then render a new custom map in a slightly different position.

Starting in 2005, Google, Microsoft, Yahoo!, and MapQuest began to offer a new class of online, interactive maps. These map services *pre-rendered* a standard set of map tiles covering the entire coverage area. A sophisticated JavaScript program running in the user's browser dynamically downloads the set of tiles that cover the user's desired map viewport. The client positions and crops those tiles on the screen to produce a map with exactly the desired size and extent.

Interactive maps have several advantages over their on-demand predecessors. Perhaps most importantly, user interaction is significantly more intuitive. Because the final step of assembling tiles into an image is done on the client, it's possible to support fluid panning of a seemingly infinitely-sized image. Pre-rendered services typically have higher quality images as well; because tiles are no-longer rendered in realtime, slow enhancements such as anti-aliased fonts can be used.

Shifting so much of the map's implementation to the client also had an unexpected effect. Soon after the release of Google Maps (the first such public service), web hackers learned to create Google Maps *mashups.* A mashup is a combination of maps with other geographically interesting data, such as those listed in the Introduction. Geographic mashups gained popularity quickly. Most major online mapping sites released official APIs that allowed web developers to create mashups "legally."

For the most part, geographical mashups so far have consisted of drawing fairly simple shapes on top of the online maps—for example, layers of pushpins (houses for sale) or polylines (bus routes). Largely ignored, however, is the practice of superimposing an entire image layer onto the underlying imagery. We speculate that this is because raster overlays are difficult to construct, as we will explore in the next section.

The difficulty in constructing raster overlays is unfortunate, because they can be quite useful. Figure 1 shows an example. The left pane shows the image of the University of California, Los Angeles as seen in either Google Maps or Microsoft Virtual Earth. The aerial imagery shows a densely built area, but the street atlas has no data describing any of the buildings or the campus' internal roads. However, UCLA publishes a detailed campus map. The right pane of Figure 1 shows the same area after we used MapCruncher to generate a raster overlay. The map can be panned and zoomed, just as was possible before the overlay was added.

The main contribution of MapCruncher is that it makes a task accessible to casual users that had typically been the domain of geographic-information-systems (GIS) professionals.

3. CHALLENGES TO THE CASUAL MAP MASHER

In this section, we consider the difficulties encountered in taking an arbitrary map—say, a PDF map of a university campus—and

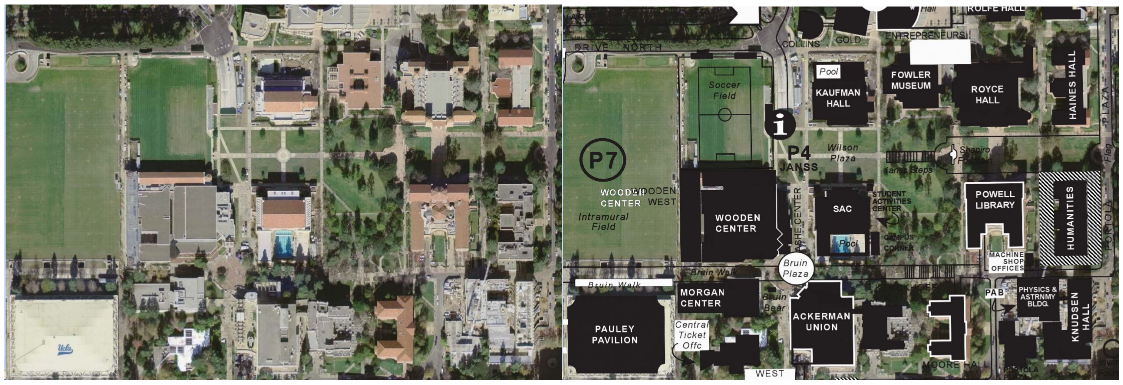


Figure 1. (*left*) Base imagery of the UCLA campus (*right*) UCLA's campus map superimposed, using MapCruncher

turning it into an interactive map layer. That is, we'd like to superimpose our map onto the road and aerial photography already provided by online mapping sites, such that the two maps can be viewed together, as in Figure 1.

Map overlays have existed for as long as maps have existed, so it may seem surprising that a new tool was necessary to accomplish a seemingly well-known task. In fact, our original intent was not to create a tool, but to create a mashup using existing tools. In this section, we describe some of the hurdles we encountered and how they motivated us to build a new tool to overcome them.

3.1 Reprojection of Unknown Map Projections

The Earth is round, but maps and the computer screens that display them are flat. Maps that depict very small extents of the Earth relative to their level of detail, such as building blueprints, can make the simplifying assumption that the Earth is as flat as the map that depicts it. However, maps of larger extent can not ignore the curvature of the Earth. A cartographer must therefore select a method to convert the position of points on the three-dimensional Earth's surface to the two-dimensional map. The mathematical functions used for this purpose are called *map projections* [20].

One spatial relationship or another is lost whenever the three-dimensional Earth is projected into a two-dimensional representation. Consequently, an astonishing variety of map projections have been invented. Each projection makes different tradeoffs, typically maintaining high fidelity in some aspect of the Earth's representation (e.g., the shape of objects) by giving up fidelity in some other aspect (e.g., apparent relative sizes of objects). Cartographers select the best projection based on a map's intended use. Most map projections are parameterized, to enable them to be fine-tuned to the location, size, and aspect ratio of the extent of the map.

For two maps to be superimposed correctly, as is our goal, they must both be drawn using the same projection. In the world of traditional GIS systems, this problem is usually easy to solve. Most spatial data comes annotated with metadata describing which projection was used to draw it, along with the projection's parameters. This information can be used to perform a mathematically exact transformation of a map into any other projection.

For casual mashups, the situation is more difficult. The vast majority of maps available on the Web have been stripped of the metadata that describes the map projection. For maps that do have metadata, it is often in a format that can not be automatically parsed—for example, a text file describing the projection in English. Consequently, it is nearly impossible to precisely or automatically reproject a typical map found on the Web.

This is a problem for a user who wishes to create an overlay. Most maps are not drawn using the same projection as is used by the major interactive online map services. Microsoft's and Google's mapping sites, for example, use the *Mercator Projection*. (Mercator is used because it is *conformal*. Conformal projections do not distort features' shapes, making it possible to overlay street maps on undistorted aerial photography.) In contrast, most other maps are not expected to be used as overlays for photographs, so instead use one of the many projections that produce less scale distortion. It is hard to guess exactly which projection a map uses by inspection because there are so many projections. For example, the USGS[1] produces maps depicting each of the 50 United States using custom projection parameters tailored to each state.

Unfortunately, this problem is not well solved by any of the numerous tools available that aid in the production of map overlays. After a week or two of tinkering with various test maps, we concluded the existing tools were all either too simple or too complex. The simple tools were limited to linear transformations such as scaling, translation and rotation. Our test maps did not use the Mercator projection, so the simple tools could not warp them sufficiently to produce good alignment at all points. The complex tools could perform arbitrary reprojections, but required complete specification of the projection, which was unavailable for our test maps.

MapCruncher addresses this problem using *approximate reprojections*. As we will see in Section 4, MapCruncher allows users to point out correspondences between the two maps, then estimates how to reproject the user's map into Mercator without a model of the source map's projection. Although less accurate than an exact reprojection, this design choice fills a useful niche in between the low- and high-end.

3.2 Management of Large Datasets

The simple, intuitive pan-and-zoom interface provided by online maps makes it easy to forget that they are providing access to immense repositories of data. Microsoft's Virtual Earth platform has nearly 200 terabytes (10^{14} bytes) of imagery. While a casual user is unlikely to ever create such a large dataset, we've found that even modest maps can overwhelm normal desktop image processing tools.

For example, consider the map of neighborhood bicycle routes produced by King County, Washington. Two of the authors commute to work by bicycle, so this map was of particular interest. We tried to overlay it on several interactive maps (Google Maps [8], Google Earth [7], and Microsoft Virtual Earth [16]) using previously existing tools. All of them required that we provide the overlay as a single rasterized image (e.g. a PNG).

The 2005 edition of the King County bicycle map is a 30"x36" poster. If rendered at a zoom level large enough that its smallest features are easily readable, it is a 3-gigapixel image. Despite considerable effort, we could not find a PDF rendering program under Windows or Linux capable of producing an output image of that size. Their failure modes were diverse and often amusing. Some ran out of RAM (3GB was available). Others filled the disk with temporary files. Some simply froze the computer.

Even if we had we succeeded in creating such a large image from our source PDF, other roadblocks would have awaited us. Similar limitations existed in the tools available both for registration of the image to a reference map and cutting it into browser-compatible 256x256 pixel tiles. Our early failure in the seemingly simple task of creating a bicycle-map overlay was among our motivations to write MapCruncher.

[1] The United States Geological Survey (USGS) is the official mapping agency for the United States.

MapCruncher was designed with enormous output images in mind. As we will describe in Section 5, our tool uses the same strategy as the large interactive map sites: instead of producing a single image, MapCruncher renders a large number of small (256x256) image tiles. This allows browsers to navigate through large custom overlays just as they do the underlying road maps and aerial photography: efficiently downloading just the sub-images they need, on-demand. In contrast, most other overlay generators that require the user download the entire overlay image before displaying any of it. This is impractical for our 3-gigapixel test map.

MapCruncher also handles large source maps gracefully generating each 256x256 tile individually, directly from just the portion of the source map that it requires. Again, this is in contrast to other tile generators that require the entire source map to be rendered in advance, even though the image may be giga- or even tera-pixels in size.

3.3 Mashing Without Programming

In the earliest days of the Web, content production was an engineering discipline. Writing HTML is similar in some ways to computer programming. Like programming, it is inaccessible to people who do not happen to be experts in the field – that is, inaccessible to most people who want to create content. Various HTML authoring tools quickly appeared, making it easier for non-experts to write web pages without needing an intimate understanding of the underlying technology.

The situation today is similar with the creation of mashups, both geographic and otherwise. They are difficult to create without first learning JavaScript, HTML, XML, esoteric APIs, map projections, and geographic coordinate systems. Our first attempt at creating a bicycle-route mashup was slowed by the requirement we learn many new disciplines, from web APIs to map projections to online maps' coordinate systems and naming schemes.

One of our motivations for writing MapCruncher was to make geographic mashups accessible to non-experts – including people who would not have been able to create a mashup without it. As we will see in Section 6, MapCruncher lets beginners create point-and-click mashups, while still allowing advanced users to customize arbitrarily.

4. APPROXIMATE REPROJECTION

In this section, we describe how MapCruncher reprojects (changes the shape of) and registers (correctly positions) the user's map such that it correctly overlays the existing Mercator-projected maps and aerial photography.

MapCruncher differs from traditional GIS systems, which generally perform mathematically exact map reprojections. GIS software usually includes a large library of commonly used projection families; the user is asked to select the one that was used to draw the source map. The user must then enter the numerical parameters that specify the exact projection. The nature of these parameters depends on the projection.

Unfortunately, as we described in Section 3.1, most maps used by our target audience have unknown projections. A user who is not a GIS expert may not know even what a projection is. Consequently, we designed MapCruncher to *estimate* the transformation. First, we ask the user to identify some landmark that can be found both on the user's map and also on the Virtual Earth map or aerial imagery; we call this identification a "correspondence." After obtaining several correspondences, we find the coefficients to a polynomial function that best fits them. A transformation with a 2nd-degree polynomial can look very similar to the transformation from many projections into Mercator.

"But wait!" a GIS professional might insist. "Polynomials may look *similar* to the right answer, but to reproject correctly, you need trigonometry. And asking for user input by pointing out map landmarks is horribly prone to error!" This is true – and the users who have spatial data annotated with all the metadata required to do an exact transformation are likely to use GIS tools, not MapCruncher. While not exact, we've found polynomials produce excellent results in a wide variety of maps. By analogy, the existence of AutoCAD does not obviate the value of Microsoft Paint.

In Section 4.1, we describe the process of gathering enough data from the user to reproject the user's map. In Section 4.2, we describe how MapCruncher uses that input to produce a usable map overlay.

4.1 Georeferencing

The first step in creating an overlay with MapCruncher is specifying a number of *correspondence points* between the user's map (the "source map") and the existing road maps and aerial photography (the "reference map"). Because the reference maps are, themselves, already registered to the Earth's coordinate system[2], each correspondence identifies the real latitude and longitude of a point on the source map.

MapCruncher provides a simple interface for specifying correspondence points. The MapCruncher GUI, shown in Figure 2, has two viewing panes. The *source pane* displays the source map, which can be panned and zoomed to arbitrary locations and zoom levels. The *reference pane* displays the reference map, using imagery from Microsoft Virtual Earth. The

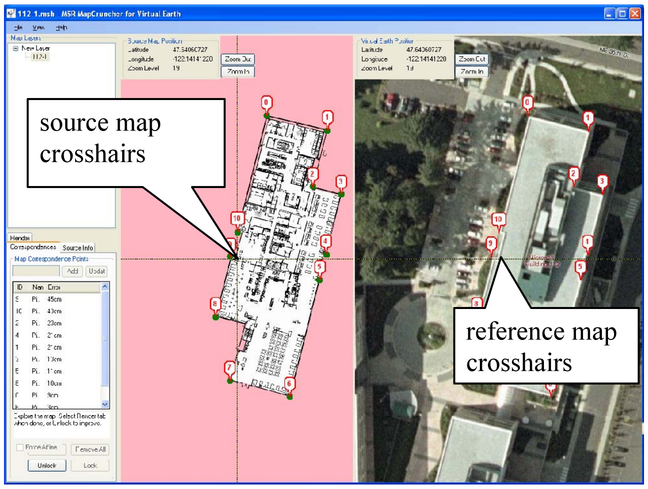


Figure 2. Establishing a correspondence between a source map and the reference map

[2] Specifically, the "WGS84" datum.

reference map can be panned and zoomed independently of the source map.

The user employs the two panes to find a location on the source map and a location on the reference map that visually correspond to each other. Any landmark that appears in both maps can be used as a correspondence. For example, in Figure 2, we are registering a building floor plan to Virtual Earth's aerial photograph of the same building. In this example, the corners of the buildings are readily visible in both views and make excellent references for a correspondence. For some source maps, it may be more convenient to use Virtual Earth's road-map view instead of aerial photography. Street intersections make excellent correspondences for many maps.

Each pane includes crosshairs that identify the center of the pane. The user indicates a location by panning the image until a feature is under the crosshairs. Once the same feature is under the crosshairs in both panes, the user clicks a button labeled "Add Point". This process is repeated until there are enough correspondences. In practice, between two and twenty are required, depending on the source map.

We discovered that in maps that cover a large geographic extent, establishing 20 correspondences can be time-consuming. To speed the process, MapCruncher can helpfully guess the spot on the reference map that corresponds to an arbitrary source map point. As soon as the first two correspondences are defined, the user can "lock" the views of the source and reference maps together. When one locked map is panned or zoomed, the other follows suit.

With just two or three points, the lock is based on a poor approximation, but it is usually good enough that it greatly assists the user in establishing additional correspondences. Using locked views, the user can zoom rapidly to a new location in the source map, and the reference map follows along. Often, only a little nudging of the (unlocked) reference map is required to find the exact matching point. Thus, the third and following correspondences in a mashup become much less tedious to define. As each new point is added, the reprojection approximation improves.

4.1.1 Error display

Sometimes, the user accidentally establishes a correspondence between points on the source and reference maps that do not actually correspond. A common instance is an "off-by-one-block" error (see Figure 3).

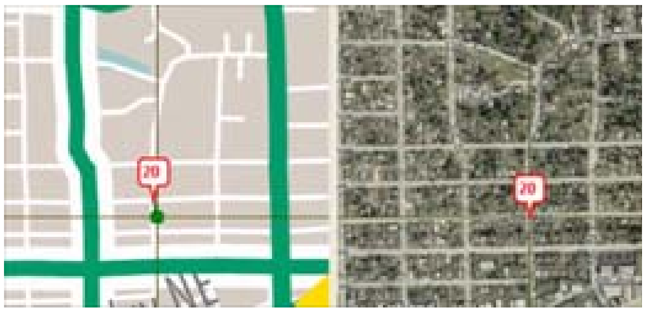
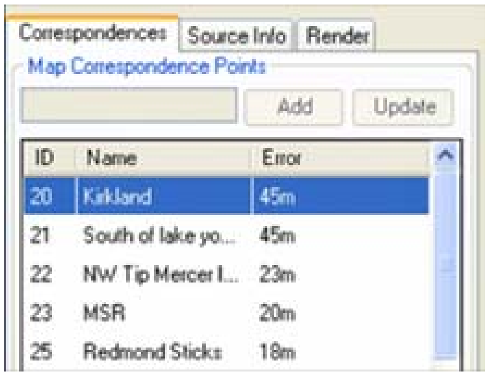


Figure 3. Establishing a correspondence between a source map and the reference map



ID	Name	Error
20	Kirkland	45m
21	South of lake yo...	45m
22	NW Tip Mercer I...	23m
23	MSR	20m
25	Redmond Sticks	18m

Figure 4. Correspondences sorted by disagreement

The reprojection process will dutifully attempt to distort the source map to satisfy the erroneous correspondence. However, if the reprojection is overconstrained and there are enough correct correspondences, the result will mostly respect the majority. MapCruncher uses the distance between the reprojected point and the user-placed point to find outliers. It computes the magnitude of disagreement for each correspondence, sorts by decreasing disagreement, and presents the list to the user (see Figure 4).

The observed amount of disagreement provides the user with a quick suggestion of which points might have been placed incorrectly. The user can then revisit the top few "suspicious" correspondences to ensure they're in the right place.

As an additional aid to the user, MapCruncher plots a vector from where the user placed a correspondence point towards where the majority suggests the point should have been placed (see Figure 5). In this case, the correct source map position (left side) corresponding to the marked reference map position (right side) is one block south of the point selected by the user. The disagreement vector points south, suggesting "perhaps the point belongs somewhere down there."

4.2 Reprojection

After the user has created correspondences, the next step is to generalize them, relating the *entire* source map to global coordinates. Mathematically, we need to produce a function that captures the relationship between image coordinates on the source map and image coordinates of the Mercator-projected reference map.

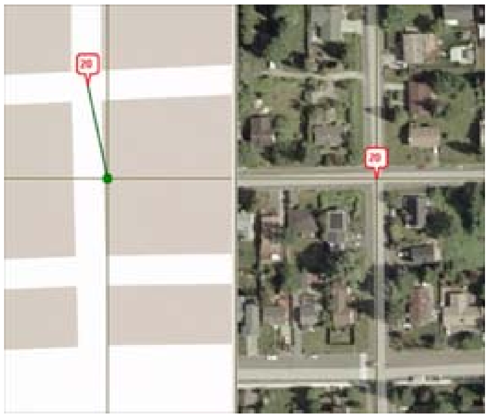


Figure 5. Disagreement vector points toward likely correct location.

The mathematically exact relationship between two maps is determined by (1) the projection of each map and (2) the parameters of that projection. The projection of the reference map and its parameters are known (in our case, Mercator). Therefore, one possible approach (which we do not employ) is to try to fit various selections of projection and parameters to the user-entered correspondence data to discover a best fit. Given the fitted model for the source map projection and the known reference projection, the function is determined.

Unfortunately, the set of projections in which source maps may be drawn is quite large, and the process of fitting parameters to each projection is diverse and involved. An alternative approach that we use in our application is to ignore the precise projections, and instead use an approximation to model the entire class of potential reprojections.

Like a projection, an approximate reprojection is a class of functions selectable by parameters. MapCruncher includes two classes of reprojections: (1) affine reprojections, including both general affine reprojections and the restricted subclass of rigid reprojections, and (2) bivariate polynomial reprojections, specifically the subclass of quadratic reprojections. These will be discussed in the following sections.

4.2.1 Affine reprojection

The affine reprojection is a linear relationship between the source and reference coordinate systems:

$$s_x = c_{00}r_x + c_{01}r_y + c_{02}$$

$$s_y = c_{10}r_x + c_{11}r_y + c_{12}$$

An advantage of the affine reprojection is that it has only six parameters, which can be inferred with as few as three correspondences. (Each correspondence provides two constraint equations, one in *x* and one in *y*.) In Section 4.2.5, we discuss how these parameters are estimated.

A limitation of affine reprojection is that it preserves straight lines. If the source map is in, for example, a conic projection, then exact reprojection will change straight lines in the source map into curved lines in the reference projection. Affine reprojection cannot produce this effect, and will therefore introduce errors into maps where this effect is noticeable.

4.2.2 Rigid reprojection

A restricted subclass of affine reprojection is rigid reprojection. A rigid reprojection constrains the affine projection to only allow translation, scaling, and rotation, eliminating asymmetric scaling and skew. If both source map and reference map obey conformal projections (a common property which is true of Mercator), then the best affine projection will always be rigid.

The advantage of a rigid reprojection is that it has only four degrees of freedom instead of six, and can thus be determined with only two user-provided correspondences rather than three.

MapCruncher includes a simple mechanism by which the implementation of affine reprojection may be reused to implement rigid reprojection. As described above, affine reprojection requires three correspondences, whereas rigid reprojection requires only two. MapCruncher synthesizes a third correspondence and uses the resulting three correspondences to solve for the affine reprojection parameters as described above. Suppose we have two correspondences A and B, each comprised of points (A_s,A_r) and (B_s,B_r) on the source and reference maps, respectively. To synthesize the third correspondence, we find on each map a point C that forms a right isosceles triangle with A and B.

4.2.3 Quadratic reprojection

To accommodate maps where the constraints of affine reprojection introduce significantly visible error, we also provide polynomial reprojection, in particular the subclass quadratic reprojection. A quadratic reprojection takes the form:

$$s_x = c_{01}r_x^2 + c_{01}r_xr_y + c_{02}r_x + c_{03}r_y^2 + c_{04}r_y + c_{05}$$

$$s_y = c_{11}r_x^2 + c_{11}r_xr_y + c_{12}r_x + c_{13}r_y^2 + c_{14}r_y + c_{15}$$

By introducing terms of higher degree than the linear terms of affine reprojection, the quadratic reprojection can better approximate an exact reprojection, including some curvature. The curvature is still not perfect, because exact reprojection generally involves trigonometric functions rather than polynomials. In practice, however, we have found that the quadratic reprojection usually suffices. For most source maps, reprojection error is dominated by sources other than the limitations of our quadratic model.

The disadvantage versus affine of quadratic reprojection is that it requires six user-entered correspondence points to completely constrain its parameters. These parameters are inferred in the same manner as those for affine reprojection, as discussed in Section 4.2.5.

4.2.4 Higher-degree polynomials

Of course, the technique used for quadratic reprojections can be extended to polynomials of higher degree. We have found in practice that quadratics are sufficient for most applications. Higher degree polynomials might better approximate the exact

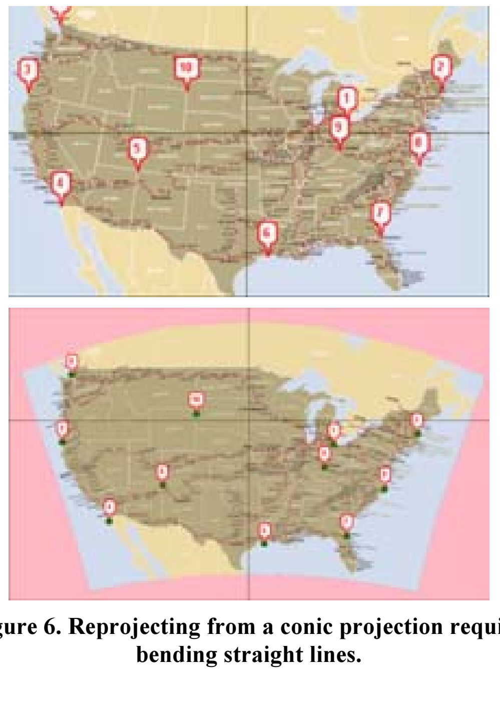


Figure 6. Reprojecting from a conic projection requires bending straight lines.

trigonometric projection for some maps where curvature is exaggerated, but we have only rarely encountered such situations.

The top image In Figure 6 shows a source map in conic projection. The bottom image shows the map reprojected into Mercator, based on eleven manually identified correspondences. Because the image covers a large longitudinal extent, its curvature is noticeable in the reprojection. Even in this extreme case, the quadratic reprojection is sufficient for the scales of interest.

4.2.5 *Parameter fitting and Error Minimization*

The preceding subsections describe formulas and their parameters, but not how the parameters are determined. If a user provides the exact number of correspondences necessary for the reprojection (three correspondences for affine or six correspondences for quadratic), the parameter values can be determined with a simple matrix inverse. The resulting reprojection will place the specified correspondence points of the reprojected map at the exact locations on the reference map that the user has identified.

A user may choose to provide more correspondence points than strictly necessary. There are several reasons for this: The user may be concerned about the possibility of errors in the source map; the user may have some uncertainty about which locations in the source map correspond to which locations in the reference map; or the user may be unsure of where points should be optimally placed to minimize distortion of the reprojected map. When additional correspondences are specified, it is not generally possible to satisfy all correspondences simultaneously. Instead, MapCruncher produces a reprojection that places the specified correspondence points of the reprojected map at locations nearby those on the reference map that the user has identified. In particular, it attempts to minimize the mean squared distance between the reprojected correspondence points and the reference points. In other words, the parameters are determined using a linear least-squares fit. In practice, our system employs singular value decomposition (SVD) [1] to implement the fitting procedure.

4.2.6 *Automatic selection*

MapCruncher can reproject a source map with as few as two correspondence points established, using rigid reprojection. When a third point is added, the application begins using a general affine reprojection. As more points are added, the approximation is improved by using parameter fitting to average out error. Once there are at least n correspondences, our application switches to a quadratic reprojection.

The minimum value of the threshold n is six, since that many correspondences are required to determine a quadratic reprojection. We chose to use n=7, because with only six points there is no redundant information, so tiny errors can cause the application to generate a quadratic projection with undesirable distortions. In contrast, the same six points overspecify an affine reprojection, providing sufficient redundancy to average out error.

MapCruncher allows the user to disable quadratic projection, in cases where affine reprojections with more than 7 correspondences are desired. This behavior is useful for source maps where it is important that straight lines not be curved, such as building floorplans.

5. TILE RENDERING

Once enough correspondences have been established, MapCruncher has sufficient information to determine the source-map pixel corresponding to every latitude and longitude covered by the source map. When the program is used interactively in the "locked" mode (i.e., the source map and reference map moving in tandem), reprojected tiles are rendered and cached on-demand each time the user looks at a new area of the world.

However, in the final mashup, we decided against on-demand rendering, for several reasons. First, rendering can take a long time. This is a particular problem for slow computers, mashups that have a large number of input source maps, and mashups that have complex PDFs as source maps. Because storage is cheap and responsiveness of web applications is important, it makes more sense for MapCruncher to exhaustively pre-render all image tiles—just like Virtual Earth and Google Maps. Second, on-demand rendering places a much higher complexity burden on the user. It would require special configuration of the web server, which is often not possible for people without administrative access to one, and difficult for beginners. On-demand rendering would also limit the number of compatible web server implementations and server operating systems. In contrast, pre-rendered tiles are just data: they can be served from any Plain Old Web Server (see Section 6.2).

For these reasons, MapCruncher allows users to a pre-render a

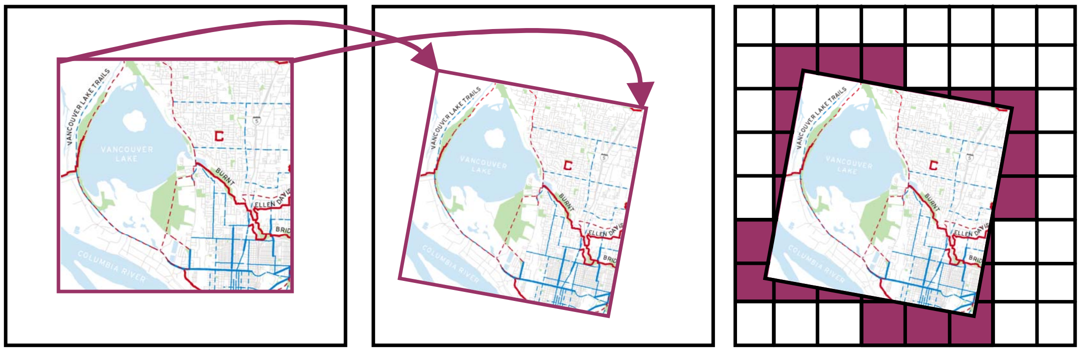

Boundaries in source coordinates are projected… into boundaries in reference coordinates… and used to select tiles which contain the region of the source map.

Figure 7. Identifying the set of tiles that cover a reprojected map

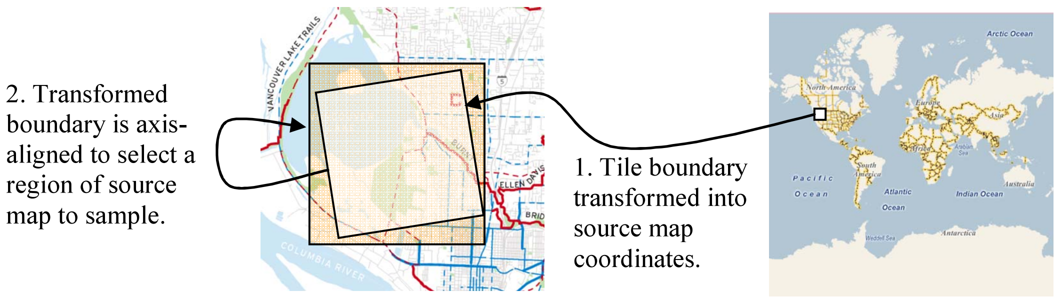

2. Transformed boundary is axis-aligned to select a region of source map to sample.
1. Tile boundary transformed into source map coordinates.

Figure 8. Identifying the set of tiles that cover a reprojected map

database of image tiles. Users can first select the maximum zoom level for which tiles are produced. Each additional zoom level increases the spatial resolution of tiles by a factor of two in each dimension, and therefore increases the total storage requirements by a factor of four.

5.1 Determining geographic extent of source map

The geographic extent of the source map is determined by applying the inverse of the reprojection function to the boundaries of the source map. The inverse function maps from source map coordinates to reference map coordinates, so this process produces a boundary in reference coordinates that corresponds to the boundary of the source map. The points on the reference boundary are converted into tile coordinates to select the set of tiles that contain the entire reprojected source image (see Figure 7). This tile selection process is repeated for each zoom level for which the user desires to output tiles.

5.2 Selecting region of source map to sample

In theory, the best-fit reprojection function is all that is needed to produce a complete set of rendered tiles: it allows us to find the source-map pixel that corresponds to every possible reference-map pixel. However, there are many choices in the implementation of tile rendering that can have dramatic effects on its efficiency and resource requirements.

There are two straightforward approaches by which rendering could be done, neither of which we use. First, one could use the reprojection function (along with information about the location and zoom level of the tile being rendered) to map each individual pixel's location to a location in the source map; render the area of the source map defined by the extent of the pixel; and use the result of the rendering to assign visibility and color to the pixel. This approach is prohibitively expensive in terms of the computational cost per pixel.

A second inefficient approach is to first render the *entire* source map at the scale dictated by the tile set's zoom level. Then, for each pixel in a final rendered tile, find the corresponding pixel in the enormous, rendered source map. This approach, used by many overlay tools, is computationally efficient and conceptually simple because the source map needs to be rendered only once. However, it is prohibitively memory-intensive when rendering maps at high zoom levels. This is because rasterizing a vector image such as a PDF source map requires memory proportional to the size of the raster. For many source maps, rasterizing the entire thing at a high zoom level can result in a giga- or tera-pixel image.

MapCruncher's approach is to render the pre-image of each tile one at a time. This approach is efficient in both computation and memory. For each final rendered tile to be generated, it determines the section of the source map needed to generate the tile, and renders *only* that part of the source map. To determine the section, the boundary of the reference tile in reference coordinates is transformed through the reprojection function to produce a boundary in the source map coordinate system (arrow 1 in the Figure 8). An axis-aligned bounding box is drawn around the transformed tile boundary (as shown in the figure). The region is axis-aligned because most source map formats are amenable to sampling in such regions. The region is also slightly enlarged to account for projections with high curvature.

Once this target region is computed, we ask the underlying PDF renderer to produce a sample image of only the portion of the source map needed to render the final tile. This is memory-efficient because it only requires rasterization of small (approx. 300x300 pixel) images. Of course, at high zoom levels, these images may cover a minute portion of the source map. MapCruncher uses a PDF renderer licensed from Foxit Software [5], which cleverly stores the list of image vectors in the PDF so that most of them can be pruned (not rendered) when viewing a tiny region, making the pre-image approach even more computationally efficient.

Finally, this small region of the rasterized source-map image is sampled to produce the final rendered tile. For each of the 256x256 pixels in the final tile, the reprojection function is used to find the four nearest pixels in the source-map image. These four pixels are combined using bilinear interpolation.

6. DEPLOYMENT

One of our guiding principles in writing MapCruncher was that it should minimize the specialized knowledge required by the user as much as practical. Therefore, it was important that MapCruncher not only create map image tiles, but automatically emit a fully working web application that gives users instant gratification of seeing their creation come alive.

6.1 Sample Web Page

When MapCruncher renders output tiles, it also creates a sample web page that shows the user's map layers overlaid on top of Virtual Earth's street maps and aerial imagery. The sample page also includes a "Find…" box, allowing users to search for businesses (using Virtual Earth's yellow pages service) and overlay pushpins right on top of their custom maps. The new "VE3D" digital globe is also supported – instantly draping the user's map tiles on top of a three-dimensional rendering of Earth that can be viewed from any position and angle. VE3D uses a digital elevation map that is compatible with MapCruncher tiles, so bicycle routes can actually be seen going up and over mountains (see Figure 9).

To some, it might seem that this sample web page is unnecessary: surely anyone who bothered to create a mashup will also bother to write their own web page to display it! By way of counterargument, consider Microsoft's basic HTML editor, FrontPage. When a user opens FrontPage, it titles the default blank document "New Page 1" – a string that appears 6 million times in the MSN Search index as of this writing.

6.2 "Plain Old Web Server" Requirement

Another important constraint in our design was that the rendered mashup can be served by a "POWS" – Plain Old Web Server. That is, we do not depend on the availability of any special server features, such as the ability to execute CGI scripts, interpret server-side includes, or configure custom error documents. Dependence on these features would limit our audience to technical users who have administrative access to a web server.

MapCruncher requires nothing from a web server other than its most basic function: return a file if it exists and a 404 error code if it does not exist. This means that users can create public mashups even without owning a web server – they can simply upload the output directory to any public web service. This includes both beginner-oriented services such as GeoCities and more advanced offerings such as Amazon S3. In both of these examples, server-side execution and custom web configuration are not available.

6.3 Applications

MapCruncher has a wide variety of uses. Three of our favorites are described here and available on the web.

6.3.1 Pacific Northwest Bicycling Guide

Our most ambitious mashup to date is the Pacific Northwest Bicycling Guide [14] – a seamless combination of bicycle route maps from 7 counties and 8 municipalities around Washington and Oregon. Overlaying bicycle maps on top of the underlying street maps is quite valuable. Bicycle maps typically do not show the smaller off-trail roads, making it difficult to plan an end-to-end trip without the overlay. The seamless integration of aerial photography can also clear up ambiguities in sometimes casually-drawn bicycle maps. For example, we used it to discover that a pedestrian overpass was available on trail not clearly depicted as crossing a major highway. The "Find a business…" feature of Virtual Earth also makes it easy to, say, find an ice cream shop along your route on a hot day.

6.3.2 National Park Service Maps

The United States' National Park Service publishes maps of more than 200 National Parks in the public domain [10]. Each is annotated with a rich set of data, including hiking trails, the names of many small lakes and rivers, geological formations, etc. In contrast, vendors of the street-map data found in most online mapping sites simply depict the park as a large blank area with the park name.

Using MapCruncher, it's easy to combine the rich annotations found in the park maps with the aerial and satellite photography provided by Virtual Earth [13]. It's also easy to leverage Virtual Earth's other features to produce new composite services – for example, getting driving directions from your home to the ranger station, drawn right on top of the park map.

6.3.3 Do-It-Yourself Aerial Photography

Virtual Earth and Google are both adding and updating imagery as quickly as they can; it's a top priority for them. However, for the foreseeable future, there will always be people who want high-quality aerial photography in areas that do not yet have coverage. Previously, there was no way for users to add their own photography. MapCruncher makes this easy for the first time.

Two members of the MapCruncher team, coincidentally, are private pilots. While on a flight 4,000 feet over the small town of Forks, Washington, we had the idea of using new aerial

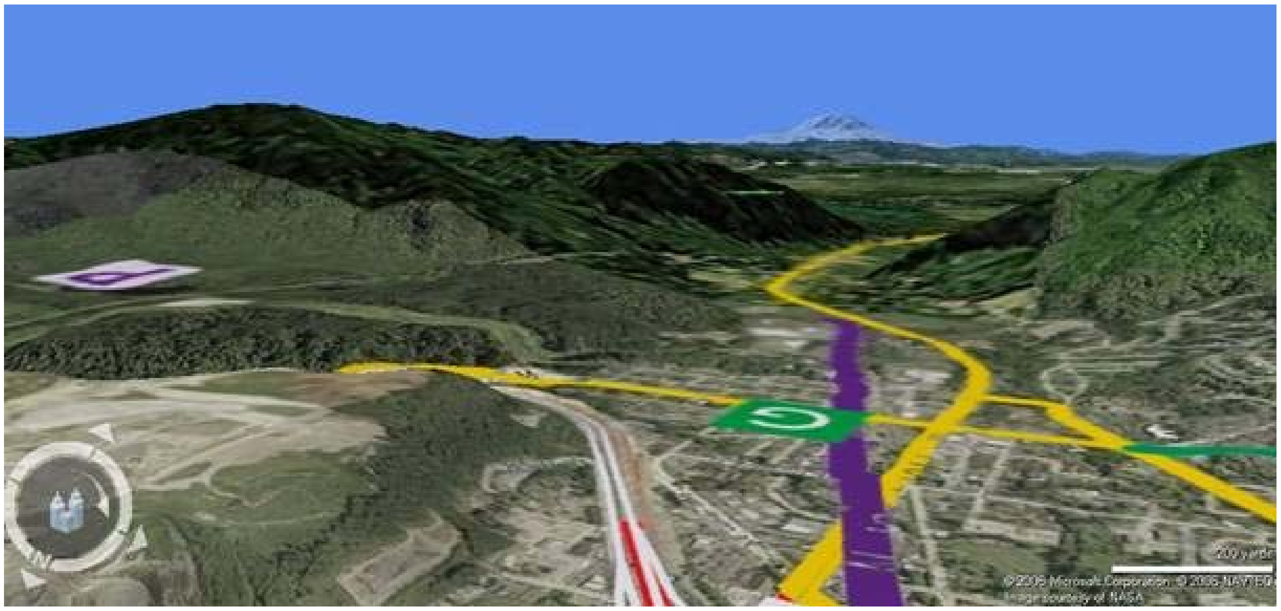


Figure 9. Bike trails on tiles emitted by MapCruncher draped over VE3D terrain.

photography as a source-image instead of a map. We circled for several minutes, taking a few snapshots out the side window with an old digital camera.

On the ground, we imported the photos into MapCruncher, using distinctive landmarks shared by both our photos and the Virtual Earth reference photos. The results were surprisingly good [12]. While seams between the images are visible, the polynomial fitting function was able to effectively ortho-rectify large portions of our photos. (Most of them had severe perspective distortion due to being shot at an oblique angle.)

Despite these problems, there was a dramatic increase in image quality, especially relative to the time and financial cost of our project. In May of 2006, Virtual Earth's coverage of Forks was 1m/pixel, 12-year old, black-and-white USGS aerial photography; Google's was 8m/pixel satellite photography. After one hour in a small airplane and a few hours on the ground, we had modern, full-color, 0.5m/pixel photography of a market so small that it's unlikely to be re-photographed by Microsoft or Google in the near future.

7. A COMPOSABLE VIRTUAL EARTH

Most of the mashups we've seen to date are interesting because the whole is greater than the sum of the parts. For example, having a bicycle map integrated with a street map is more useful than either one individually. To get the most utility from mashups, it's not enough to combine users' maps with Virtual Earth. We also need a way to make them easily composable *with each other*.

Ideally, mashups will no longer be thought of as individual sites, disconnected from the rest of the world. Instead, the building blocks of mashups—the layers of rasters, points, and lines that underlie them—should be composable, interchangable building blocks. We envision a world where mashups have more structure, so that the bicycle layer we render can easily import the Doppler weather data you've rendered, and can *be* imported into the web site that features hiking layers. If people publis their applications *and* the underlying data in a semantically meaningful way, a nearly infinite set of innovative and diverse applications are sure to follow.

MapCruncher tries to take a step in this direction by cleanly separating the imperative code that run the mashup from *declarative* code that describes the raster layer being imported. Specifically, each time MapCruncher renders tiles, it also describes those tiles—their geographic position, rendering depth, and so forth—in an XML file specially seeded with strings that can be found by search engines. With enough people creating MapCruncher layers, we can collectively create an enormous database of interesting data layers, all geographically registered to compatible coordinate systems and instantly searchable using existing search engines.

Who knows what kind of interesting mega-mashups might follow?

8. ACKNOWLEDGMENTS

The authors would like to extend their sincere thanks to Danyel Fisher, Steve Lombardi, Karen Luecking, Joe Schwartz, Chandu Thota, and the many testers who provided us valuable feedback.

9. REFERENCES

[1] H. Abdi. "Singular Value Decomposition (SVD) and Generalized Singular Value Decomposition (GSVD)." In N.J. Salkind (Ed.): Encyclopedia of Measurement and Statistics. Thousand Oaks, Oct 2006.

[2] chicagocrime.org, http://www.chicagocrime.org/map/.

[3] W. K. Edwards, A. LaMarca. Balancing Generality and Specificity in Document Management Systems, Interact'99.

[4] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, O. Shivers. The Flux OSKit: A Substrate for OS and Language Research, 16th SOSP, Oct 1997.

[5] Foxit Software, http://www.foxitsoftware.com/.

[6] GeoRSS. Graphically Encoded Objects for RSS feeds, http://www.georss.org/.

[7] Google. Google Earth, http://earth.google.com/.

[8] Google. Google Maps, http://maps.google.com/.

[9] J. S. Heidemann, G. J. Popek. File-System Development with Stackable Layers, ACM TOCS 12 (1), Feb 1994.

[10] Harpers Ferry Center. National Parks Service Maps, http://www.nps.gov/carto/.

[11] housingmaps.com, http://www.housingmaps.com/.

[12] MapCruncher team. Do-It-Yourself Aerial Photography, http://research.microsoft.com/mapcruncher/Gallery/Forks/

[13] MapCruncher team. National Park Maps, http://research.microsoft.com/mapcruncher/Gallery/National Parks/

[14] MapCruncher team. Pacific Northwest Bicycling Guide, http://research.microsoft.com/mapcruncher/Gallery/NWBike/

[15] N. C. Hutchinson, L. L. Peterson. The x-Kernel: an Architecture for Implementing Network Protocols, IEEE Transactions on Software Engineering 17 (1), pp. 64-76, Jan 1991.

[16] Microsoft. Microsoft Virtual Earth, http://www.microsoft.com/virtualearth/default.mspx.

[17] Open Geospatial Consortium. Geography Markup Language, version 3.1.1.

[18] RunwayFinder – a flight planning tool for pilots, http://www.runwayfinder.com/.

[19] Seattle Bus Monster, http://www.busmonster.com/.

[20] J. Snyder. Map Projections-A Working Manual, United States Government Printing, Feb 1983.