Towards gnuGIS: Version 0.1

Adrian Custer, UC Berkeley

May 11, 2000

Contents

1 Preface **3**

2 Introduction **4**
2.1 A brief history of geographic analysis 4
2.2 Organization of this discussion 4

3 The gnuGIS Object System **5**
3.1 The base object types 5
3.1.1 User objects 5
3.1.2 Geometric objects 6
3.1.3 Spatial primitives? 7
3.2 Memory structures 7
3.2.1 Display structures 7
3.2.2 Query structures 8
3.3 Is this love baby, or is it just—confusion? 8

4 The gnuGIS Object Manipulation System **9**
4.1 Creation, import and modification of spatial data 9
4.1.1 Data Creation 9
4.2 Display system 9
4.3 Analysis system 10
4.4 Storage 10
4.5 Manipulation 10
4.6 Output 10

5 Standard gnuGIS Data **11**

6 Process **12**
6.1 The viewer 12

7 Conclusion **13**

1 Preface

The free software community has started a number of projects related to Geographic Information Systems (GIS) including libraries, viewers and entire systems. The development of a modern, free GIS is within reach if the recent work can be leveraged effectively. This work is an attempt to discuss these projects in the context of the design of a free GIS (called gnuGIS in the following discussion). The discussion presents the design of the system, addresses the re-use of existing code (either applications or libraries) and plans the software development process. This is a work in progress which will hopefully benefit from the discussion it generates.

The effort to create a free GIS should aim to create a system which is advanced, which can be extended for different purposes and which will be upgraded over time. Therefore gnuGIS is conceived as based on object data types, as explicitly facilitating spatial analysis, as being made up of replaceable (or upgradable) software components and as being bundled with a free worldwide data set.

I enter this discussion from a user's perspective. I am an ecologist not a programmer. However, I have been spending a lot of effort trying to figure out how ecologists can use GIS for biological analysis. Like many, I have found that most GIS is full of promise but unable to perform much analysis. Many disciplines have undertaken spatial analysis, each in its own way (*e.g.* statistics—point pattern analysis; geology—trend surface analysis; landscape architecture—overlays and buffers; landscape ecology—shape indices; ecological modeling—cellular movement models or 'metapopulation' models). I have learned what I could about programming in order to better understand how a well designed GIS could be used for ecological analysis. This is just to say that those of you who understand better the programming issues need to flesh out the discussions of data structures and algorithms and define the separation of the various software components.

2 Introduction

2.1 A brief history of geographic analysis

(To be done later as justification for my analytical kick.)

2.2 Organization of this discussion

The best way to undertake this discussion is to break apart a geographic information system into its component pieces and address each in turn. At the most basic level, a GIS is a way to input or create, display, analyze, store, manipulate and output spatial data objects.

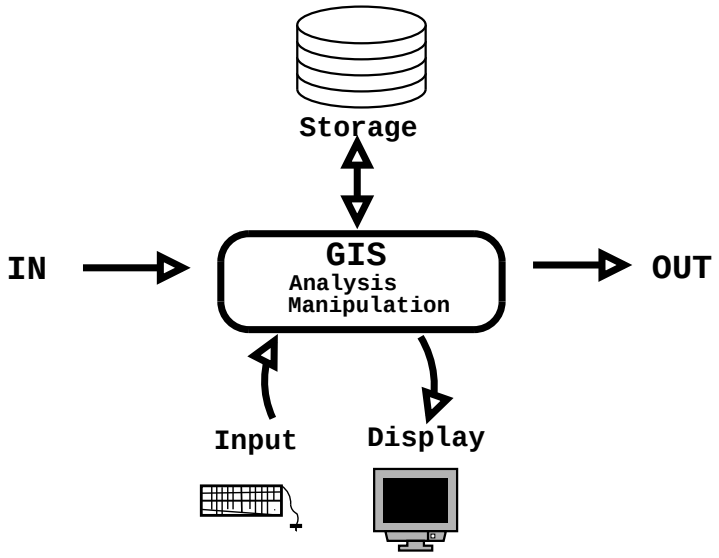

Figure 1: The components of a Geographic Information System

The foundation of an information system is its representation of the data objects it manipulates. The discussion therefore starts with a presentation of the gnuGIS object system. The capabilities of the GIS emerge from its ability to manipulate its object types. The discussion will continue by addressing each sub-system in turn.

3 The gnuGIS Object System

The object system is obviously the core of the matter. As I write this, I can see that there is much I do not yet understand and that this is exceedingly tough. This discussion needs some good programmers to review this and explain better the components such as the trade-offs of various tree structures.

The history of GIS is probably well know to everyone reading this. The notable evolution has been in the objects capable of analysis: the vector/raster split, the point, line, polygon trichotomy and recently the appearance of object-GIS (as in the proprietary Arcinfo8). Object types can subsume the previous data structures and be expanded to new types such as explicit mathematical fields (not raster representations of the field but the actual thing) or river networks (with properties such as always flowing downhill, increasing water flows and watershed definition of a landscape).

3.1 The base object types

The foundation of the object system is a user data type. This in turn contains other types: a meta-information type, an attribute type, a constraint type and a container type which can contain any combination of user types and geometric types. The geometric types might be constructed from primitives for speed of rendering. The terminology I am using is a mess. I use 'type' here as meaning a kind of object derived from a particular object hierarchy. I'm quite happy to change this term if good suggestions arise..

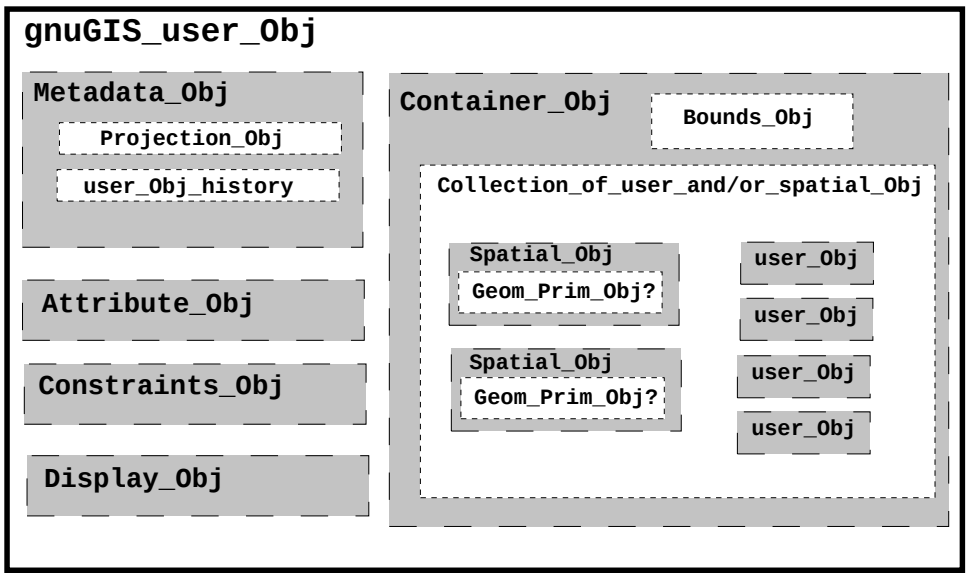

Figure 2: Object Structure of gnuGIS

3.1.1 User objects

The user object type serves two functions. It can be the atomic data element or a collection of other instances of itself.

Atomic form user objects In its atomic form, the user object represents individual pieces of the environment for example stuff like rivers, and roads and forests and elephants and smoke plumes

and ... This atom contains information about itself—a metadata object which tracks its projection and history. The atom also contains attributes of itself in an attribute object. The atom contains a constraint object which can constrain attribute values or the combination of the atom with other atoms of known type. For example, a house type could constrain itself to generate an error if it were to overlap the x,y co-ordinates of a water object. I am not sure that this is the best way to deal with constraints. Finally, the atom contains a spatial type.

The spatial type of an atomic object consists of a geometric type. In the case of a vegetation 'polygon', the vegetation object is being represented spatially by a polygon object that is an instantiation of a polygon type. Note that it is worth considering the possibility that a user object might have different geometric representations at different times. For instance, I may get money to do 100 point vegetation samples of an area one year, then I get some more money to do some transects and a few years later I get to do an areal survey and classification. My user type is the area vegetation but my sampling scheme is different. An obvious analysis then is to see if the three sampling schemes agree about the vegetation or if they are different. In current GIS this is very hard (if even possible)-it shouldn't be.

User objects have display properties which might vary with scale or based on the presence of other objects. One might want to represent an object visually in different ways. A dot object could be displayed as a point or as a 95% probability contour (say for GPS positions). Interestingly one might want to change the display property depending on the scale resolution of the view at that moment (generalization). User objects could also have different properties for different types of analysis. For example, spatial error could be ignored for most analyses but included when it seemed critical. A user object's attributes could be constrained to hold only certain values so that a proportion would be forced to be between 0 and 1. By default gnuGIS would have a bunch of defined object types but the user could derive new types.

Container form user objects User objects could act as containers of other instances of user objects. The idea is to allow for an object that holds a number of polygon objects and ensures that the whole is consistent. For example, consider the classic polygon coverage or layer. A similar container would require that all objects it contains are of the same class (polygon like), have consistent attribute data layers, do not overlap and yet occupy the whole area, and share a common projection system. Another example would be a road network container which would contain line objects with specifically defined connections between them. A river network similarly could be a predefined container.

The container idea is quite flexible and could even be extended to the creation of a raster-like container — a collection of regularly spaced point data. This might be too slow to be of use but might allow much easier analysis of rasterish layers or between rasterish and vectorish layers.

3.1.2 Geometric objects

All geometric objects would derive from a base generic object which should have the capability of being stored, displayed and reprojected. All co-ordinates should be four dimensional—3 space and one time with z and t defaulting to reasonable values. Also geometric objects would contain some representation of a minimum bounding volume. This is discussed in greater detail in the section on search structures, section 3.2. When a user object is given its geometric representation, the user object should keep a record of the associated projection information. Derived types would include classes for the classic point, line, area trichotomy but one could also create a volume class or even a field class, plume class or other esoteric types. The creation of these new classes must include an understanding of their behaviour in standard queries and object manipulations (buffers, overlays ...).

3.1.3 Spatial primitives?

I am not sure if one would want to define the geometric objects in terms of more primitive pieces such as points, straight lines (2 linked points), a triangle (3 lines), a triangular pyramid (4 linked triangles). I suspect that breaking apart a polygon shape into triangles would make it much quicker to render and shade using modern display technology. This is, however, just a guess based on the selling of graphics cards based on how many triangles a second they can calculate. Anyone who understands OpenGL can explain the tradeoffs in terms of how a view is sub-selected from a 3D object. Does the GIS have to have the entire scene in memory in order to render part of it (say a highly zoomed in area)?.

This is by no means a comprehensive description of the object system but seems like a good starting point for discussion. I might be totally crazy to think it's possible to build a system this way but this seems to allow me to build types capable of responding to some really wacky queries. Next I move from the objects themselves to the structures used to organize them.

3.2 Memory structures

Individual objects or containers of objects must be accessible for display and for queries. This requires that the objects (or references to them) be organized into structures searchable by the objects' spatial properties. This has to be rapid for display but should also be capable of more complex operations during analyses. I have trouble distinguishing between the needs of the display subcomponent and the storage system. These are presented separately.

The objects and containers must be searchable based on spatial information.

3.2.1 Display structures

The fundamental display operations are to sub-select (zoom in and out) or to pan (when zoomed in on an area). Speed is somewhat critical (hardware tends to make up for inefficient software design so the consideration really ought to be between speed today vs. extensibility tomorrow). Since gnuGIS has three spatial dimensions and one temporal dimension, it should be possible to create arbitrary fields of view (an image from any point in any direction any way up. Ideally, successive frames could then be created (and then stored in a movie format) where the field of view moved sequentially in space or in time in fixed increments! Not something that would be done right away but a capability to be planned for. (PNI-planned, not implemented)

As I stated earlier, these operations may be taken care of directly by the OpenGL libraries. Someone with a better understanding of OpenGL or of GUI display structures like the Gnome (soon GTK+) canvas should address this issue. Two considerations:

- if OpenGL takes care of this does that require that all the spatial properties of the user objects be held in memory, even when the display area is much smaller than the extent of the data set? Does this prevent the use of a massively large data set?
- If one wanted objects to look different at different scales, does one have to regenerate the whole OpenGL structure when one pans in or can a city object simply be changed from a dot to a blob to neighborhoods as one zooms in?

If display selection cannot be dealt with by OpenGL, then the issue is to create a structure that can select objects in an arbitrary field of view. I believe gnuGIS objects would therefore have to be stored based on their minimum bounding sphere. This outline type is less efficient than minimum

bounding rectangles but it allows the selection of objects from an arbitrary perspective[1]. The storage type would be a reactive tree of some kind. Where a large data layer is sub-selected, it should be possible rapidly to duplicate a section of the tree, re-balance it on the middle coordinate and search and display based on this structure. Anyone really knowledgeable about algorithms? Peter van Oosterom has a lengthy discussion of trees used for searching in his book *Reactive Data Structures for Geographic Information Systems* but he limits discussion to just 2 dimensions.

3.2.2 Query structures

Queries can get exceedingly complex but can be separated into three kinds: spatial queries, metadata queries and attribute queries. The latter two types of queries are straightforward types of searches which have been well implemented by databases. Spatial queries are less current so that is what I discuss here.

The simplest spatial queries I have just address above in the discussion of display structures. Since I advocate the creation of a separate viewer component, this structure should also be capable of handling simple spatial queries: presence/absence, adjacency, proximity (for buffers). Where analysis requires a more complex structure, the analysis component can build one. Analysis can be done at length of times vastly slower than re-display and still be useful. Alternatively, where a system is being set up for multiple rapid analyses, the container layer for the data can be required to maintain a complex search structure to support analysis. For instance, a river or road network can be queried for topological issues. It seems reasonable that the container that is guaranteeing consistency for the network, also be required to maintain the topological search structure.

3.3 Is this love baby, or is it just—confusion?

Okay, so I'm confused about where the search structures have to hang out. That's partially because the display system needs one, the object retrieval system needs one, the query system needs one, the analysis system needs one and the object manipulation needs one but they do not all have identical needs!

I'm also confused about time properties. If objects are to be allowed to move through space through time (say glaciers) it seems that the representation of the object has time varying limits. Where do they go?

[1] As I now understand things, minimum bounding rectangles only really make sense where you are selecting objects from an object layer of n-dimensions, where the selection space is rectangular along the axes of the bounding rectangles and where the point of view is in the nth+1 dimension. This is related to why all north arrows in ESRI software point to the top of the page. They have not allowed selection from an arbitrary point of view.

4 The gnuGIS Object Manipulation System

The gnuGIS object manipulation system can be dissected into a number of separate sub-systems. These are not mutually exclusive but provide a convenient way to discuss the functionality which is required of gnuGIS for it to be useful.

4.1 Creation, import and modification of spatial data

For gnuGIS to be of use it must be able to acquire data. Users *must* be able to create or modify objects by hand. While the object system laid out above makes data creation difficult, this is offset by the increased functionality of the system. The input of data can be relatively straight forward since documented file formats can be dumped into pre-made objects and standard containers.

4.1.1 Data Creation

User data object types must be creatable from keyboard (or mouse) input, through scripts or from plain text files. Ideally keyboard input could be done through an interface that would present a list of available objects and the fields that must be filled out. Values should be checked during import with sensible default values. For instance, where there is no elevation value, the data should be assumed to be 2-D and z assigned either to the elevation of the reference ellipsoid or to the value 0. Script or flat file interfaces should be defined so that users can import data from arbitrary sources.

Ideally, the user could create new data types and containers from within the program itself to accommodate a new data type.. This capability would undoubtedly be further down the road and since we have the full source code, this is not as important for us as it is to a commercial equivalent system! ⌣ Yeah, free software saves work!

Data may need to be streamed in from a modeling program or from an external source such as live feeds from sensor arrays. This should be planned for but not implemented (PNI).

Data import would simply be a matter of writing scripts capable of parsing other file formats and dropping the data into appropriate objects. Alternatively, a library such as shapelib could be called for this work.

I do not know of any digitizers with Linux drivers. The interface could simply be the ability to import a file generated by a digitizer program or could be more complex. The more involved capability should wait for other areas of the program to become functional.

4.2 Display system

The display system should be capable of showing data in arbitrary ways, panning or rotating the field of view, zooming in or out and displaying a series of views (like a fly over or a movie). User objects should also be representable as spatially populated shapes. This means that data objects could be shown as icons of differing sizes depending on an attribute or user objects could be shown as charts overlain on a map. This simply means that the representation method can be changed from a map like display to another kind of display which would appear in the right areas.

The display system must also be capable of responding to user queries of the display. What object is this ¡click¿? what are its attributes? What is the distance between here ¡click¿ and there ¡click¿?

4.3 Analysis system

This is a separate component (or multiple components) which I will present later because this is the core interest for me so I have lots to say.

4.4 Storage

I foresee two kinds of storage. Smaller data sets which are to be created once and then simply used for analysis would be stored in an xml flat file. More dynamic data, for instance objects whose attributes were constantly being updated externally, would be stored in a database.

The xml files would be able to store objects, collections of objects and other information such as views on these objects. In ESRI terms this means that the single xml file would be able to store all the kinds of shapefiles, index files, dbf files and project files.

Interface to an object capable database. I don't really know the difference between an object-relational and an object-oriented database so it seems to me that the former is fine for our needs. Data should be retrievable based on attribute data, arbitrary naming schemes, input time, metadata information and spatial extent. We can flesh out what this would actually mean at another point.

4.5 Manipulation

Everything from the creation of new object types, the instantiation of objects, the re-projection of objects, the association of objects into new layers, the combination of objects (intersections, combinations..) and so on.

4.6 Output

Display output is dealt with elsewhere.

Output of object data should be possible into flat files (xml, delimited, database tables). It should also be possible to export these data into standard file formats. This get difficult in that the gnuGIS object system is richer than many standard formats. Essentially this just means that extra data gets tossed (like all the constraints information.

Output of views and results as images, graphs and plots must be supported.

5 Standard gnuGIS Data

Ideally, I would like to compile and distribute an unencumbered (free speech) data set along with the software. It seems to me evident that the world could use some reference maps to facilitate discussion. On the other hand maps are political statements and should be considered that way.

The Digital Chart of the World in its vector product format incarnation is probably the best currently available base map that could be included. On the whole the data seem decent except for roads and some political boundaries. I would be tempted to exclude both layers for political reasons but that would undoubtedly be foolish. While this data set is not unencumbered, we could use it until the British, Australian or other governments with copyrighted information in it, made a big fuss. I suspect that none of them care any longer since data at much higher resolutions is now available.

The dem from the space shuttle topography mission when it is released promises to be an amazing data set. This would be too big to include in a distribution but a script access to it should be possible. The data set will cover the world at a spatial resolution possibly as good as 30 meters. It depends on what our military decides we are allowed to have!

A raster coverage of the world should shortly be provided by MODIS. Again this might be too big to include in the distribution but should be accessible in a standard way. Certainly the land classification layer could be grabbed.

Other ideas would be great.

6 Process

My current vision for the gnuGIS system would be to develop three separate systems: a lightweight viewer, a pluggable analysis system and a heavyweight creation, modification and interface system. Figure 3 shows the separate pieces of the gnuGIS system. These would be modular subcomponents of the software from a developer's standpoint and come in three separate components from a user's standpoint.

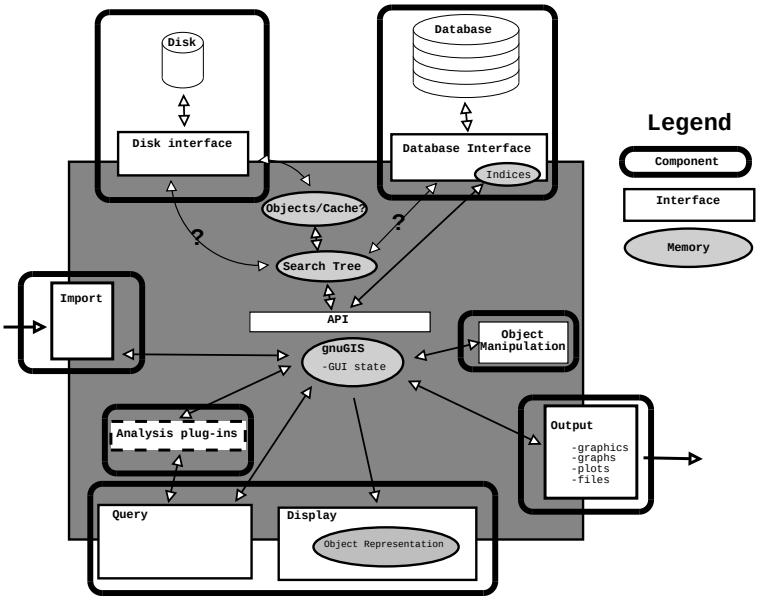

Figure 3: Subsystems of gnuGIS-Take 2

The viewer would have the display and query facilities, the output facilities and the disk interface capability. The object manipulation module would have the database interface, the data import, creation and modification pieces. The analysis component would be separate and the core component would be loaded by default.

This is just an example breakdown—totally arbitrary but a starting point for discussion.

6.1 The viewer

The lightweight viewer would essentially be able to view gnuGIS objects, to perform simple queries on the attributes and to explore the data. It would also contain the printing capabilities. The idea is that one could send data along with the viewer to someone else and they could see result views and play around with the results of an analysis. Ideally, the file would contain a set of views with associated text describing the view.

This would be akin to the early versions of Arcview or to ArcExplorer both ESRI products. Note that commercial vendors break up their software for commercial reasons so that the pieces end up replicating a lot of functionality. The idea here is to have separate components to save memory space, to allow data to be sent to someone along with the capability to do simple analysis and to modularize the software. The display system would therefore contain a spatial viewer, a viewer of object attributes and a query subsystem.

The display system should allow the user to re-display data in multiple ways, to perform queries and to see the attributes of various objects. The capabilities of the system would be quite limited but be better than simply being a map.

7 Conclusion

It seems that the viewer is worth focusing on first with more complex pieces addressed later. The object system and query structure need a lot of conceptual work but this is a start.