Seminar series nr 132

The effects of managed ruminants grazing on the global carbon cycle and greenhouse gas forcing

Jakob Lagerstedt

2006
Geobiosphere Science Centre
Physical Geography and Ecosystems Analysis
Lund University
Sölvegatan 12
S-223 62 Lund
Sweden

The effects of managed ruminants grazing on the global carbon cycle and greenhouse gas forcing

---

Jakob Lagerstedt

Degree-thesis for Bachelor of Science in Physical geography Lund University 2006

Supervisor:
Thomas Hickler, PhD

Department of Physical Geography and Ecosystem Analysis
Lund University, 2006, Nr 132

Preface

Upon undertaking what was to become this thesis, I had two mistaken beliefs to start. The first was a statement uttered upon completion of my masters thesis a few years ago - “I will never write another thesis… ever… again”. The second was that it would only take 10 weeks. Both turned out to be wrong. A thesis has been written, even though it has also taken far more time then I could have imagined. Nonetheless the result is what you hold in your hand (or read on your computer screen if you are so inclined).

As appropriate I would like to take the time to thank certain people who have been instrumental in getting to the point we are now.

My family and friends of course for their financial and other support, some of which I can not measure or put into words as I most likely have no idea of what has been going on behind the scenes at many times.

But in the end, primarily, gratitude must be aimed at my supervisor Dr. Thomas Hickler. He has had the patience to allow me the opportunity gaze into his exciting and fascinating research. Also his wisdom and insightfulness has amazed me time and time again through the process. Much like a super villain in a bad movie he always managed to be one step ahead and above all I would like to stress his philanthropic allocation of time, without which I would surly never have gotten to this point. I have had many supervisors for many academical projects, but none thus far have approached a student with the determination and generosity I enjoyed in Thomas. I wish all students had such pleasurable writing partners, however, sadly I know first hand Thomas is more an exception than the rule.

Finally I want to thank the persons who have taken some time to read parts of this paper in the never-ending quest for errors. You know whom you are and that your support has been appreciated. However, any remaining errors and omissions are mine to carry alone.

Best regards J

Abstract

Carbon is central to our existence. It plays a fundamental role in the carbon cycle. Atmospheric concentrations of carbon dioxide (CO_2) and methane (CH_4) have been increasing for about two centuries as a result of human activities. The need to understand how carbon cycles through the Earth-system is important for a variety of reasons, but mainly in order to assess the effects of these rising greenhouse gas concentrations and to project future concentrations and effects.

The world population of larger domestic grazing animals (i.e. cattle, goats and sheep) totaled around 2 billion in 1996, and has steadily increase since the 1970s by about 30%. Due in part to their large numbers the worlds grazing animals are thought to have an affect on the global carbon cycle.

Many models exist that produce estimates of the carbon cycle and its interaction with future climate change. One well-documented and reviewed model is termed the LPJ-DGVM ecosystem model. However it does not in its current state account for domestic grazing animals.

Therefore the focus of this thesis will be an attempt to develop a method for incorporating the effects of grazing animals into the LPJ-DGVM model. Further the method will be used to test the thesis statement “grazing herbivores exhibit a measurable impact on the carbon cycle and merits inclusion in carbon cycle models”.

Results suggest that, indeed, the worlds grazing animals can have an effect on the global carbon cycle and merit inclusion in carbon cycle models. Results for global NPP fluxes show lower productivity over the studied period of about 5%, which follows the same trend as other studies. Results for global NEE fluxes are surprising and suggest the biosphere is acting as a 50% stronger sink when grazing is taken into account. At present no satisfactory explanation for the somewhat contradictory NEE results has been found. Although a unverified theory suggests a conceptual flaw in the LPJ-DVGM models treatment of grassland fires. Nonetheless contradictory results are common in this line of research, especially in the early stages, and a conclusive search for the mistake can take months before it yields a valid result.

Keywords; Geography, Physical geography, Carbon cycle, Ecological modeling, Ruminants, LPJ-DGVM, C++

Table of contents

1 Introduction

Carbon is an elemental part of life on the Earth. It plays a fundamental role in the structure, chemistry, and nutrition of all living cells. And life plays an principal role in the carbon cycle. To illustrate the statement above please consider the following scenario of carbon exchange to and from the biosphere;

Autotrophs are organisms that by the use of an external source of energy (most often solar radiation) produce their own organic compounds using carbon dioxide from the air or water in which they live. Thus they sequester carbon from the atmosphere into the biospehere. Later carbon is transferred within the biosphere as heterotrophs feed on autotrophs. For example by means of uptake of dead organic material (detritus) by fungi and bacteria. After this most of the carbon leaves the biosphere again through respiration.

The above is however only one example of carbon transference and many more are known to exist. Another example can be carbon leaving the biosphere when dead organic matter (such as peat) becomes incorporated in the Geosphere through various processes.

The global carbon cycle (as depicted in figure 1) is the theory explaining the annual cycle by which carbon is exchanged (Chapin et al 2002).

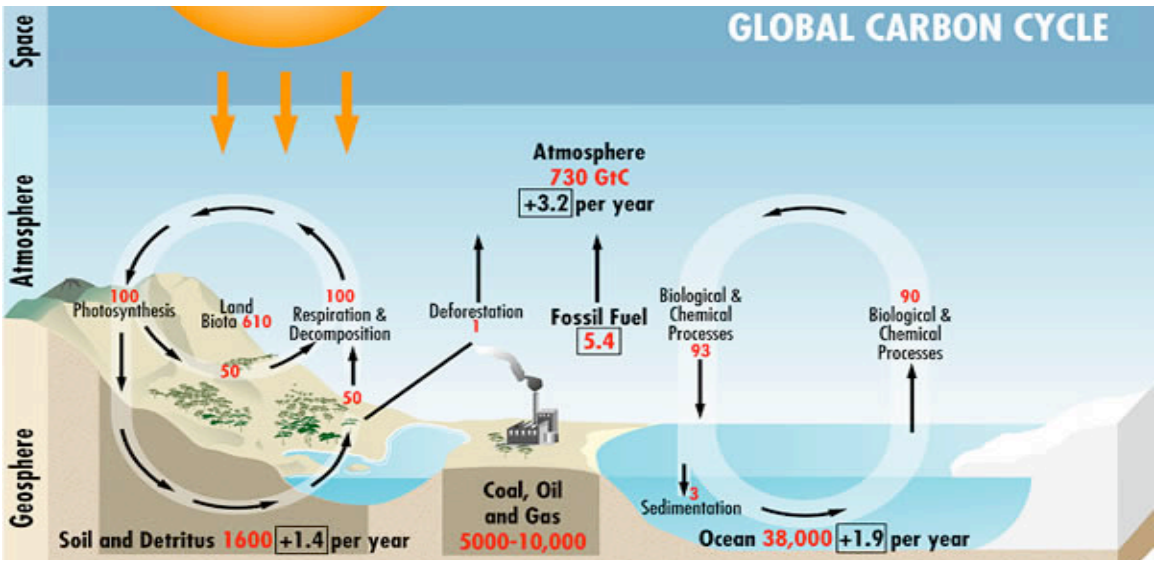

Figure 1 Diagram of the carbon cycle. The red numbers indicate how much carbon is stored in or exchanged between various reservoirs, in billions of tons (GtC). The black numbers in boxes indicate how much each carbon pool has increased, calculated over the period 1990 to 1999, in billions of tons carbon. The values shown are approximate and considerable uncertainties exist as to some of the flow values. (www.bom.gov.au/info/climate/change/gallery/images/9.jpg (accessed November 31st 2005))

Various models exist that attempt to simulate the global carbon exchange. One such model LPJ-DGVM (Smith et al. 2001, Sitch et al. 2003, updated hydrology by Gerten, D. et al. 2004) is a dynamic ecosystem model that simulates ecosystem processes and vegetation distributions and dynamics. LPJ-DGVM simulates the global carbon cycle through a well-established and evaluated ecosystem model. However, unlike some other models it has yet to account for herbivores grazing and its impact on the global carbon cycle.

The world population of larger domestic grazing animals (i.e. cattle, goats and sheep) totaled around 2 billion in 1996, and has steadily increase since the 1970s by about 30%. The worlds grazing animals primarily affect the global carbon cycle in three ways:

1. Through the long-term or permanent removal of forest cover and conversion to a non-forested land use.
2. During consumption of vegetation carbon is removed from the grass plants biomass. Only a minor portion of this is returned to the biomass as feces.
3. As a consequence of ruminant metabolism methane is added to the atmosphere.

When one considers the population numbers and the modes of interaction it is suggested that ruminants grazing systems could have an impact on the global carbon cycle. Therefore the focus of this thesis will be an attempt to develop a method for incorporating the effects of grazing animals into the LPJ-DGVM model. Further the method will be used to test the thesis statement "grazing herbivores exhibit a measurable impact on the carbon cycle and merits inclusion in carbon cycle models".

1.1 The global carbon cycle

The carbon cycle is the biogeochemical cycle by which carbon is exchanged between the biosphere, geosphere, hydrosphere and atmosphere of the Earth. It is often depicted as the exchange between four pools or reservoirs and their interconnected pathways of exchange. The pools are the atmosphere, terrestrial biosphere, oceans, and sediments. The carbon exchanges between pools occur because of various chemical, physical, geological, and biological processes. Some of these processes are quicker and some are slower in their action. (Chapin et al 2002)

Carbon in the atmosphere subsists mainly as the gas carbon dioxide (CO_2). Even though it only constitutes approximately 0.04% of the atmosphere overall, it is considered important in supporting life. CO_2 among with other gases such as methane and chlorofluorocarbons (CFC) are all greenhouse gases whose concentration in the atmosphere has been increasing in recent decades, contributing to global warming. Carbon exchange in the atmosphere is a complex process and to illustrate this a few examples of pathways are given below (Cox et al 2000);

Carbon is taken from the atmosphere in several ways:

- By the process of photosynthesis, green plants absorb solar energy and remove CO_2 from the atmosphere to produce carbohydrates (sugars). This process is most dynamic in relatively new forests where tree growth is still rapid.

- The presence of plant life intensifies the weathering of soils, leading to the long-term but slow uptake of CO_2 from the atmosphere. In the oceans, some of the carbon taken up by phytoplankton to make shells will after they die settle to the bottom to form sediments.

- The solubility of carbon dioxide in water is a strong inverse function of temperature (i.e. greater solubility in cooler water). Thus the cooler surface seawater near the poles is able to dissolve CO_2 in the deep ocean interior through a process referred to as the solubility pump.

(Above Berner 2003)

The fixed carbon can be released back to the atmosphere in many different ways, some of which are listed below.

- Plants and animals consume carbohydrates through the reverse of process photosynthesis often referred to as respiration. This is an exothermic reaction that involves the breaking down of carbohydrates into CO_2 and water.

- The process known as decomposition entails a respiration that consumes organic matter by bacteria and fungi. Fungi and bacteria break down the carbon compounds in dead animals and plants and convert the carbon to CO_2 if oxygen is present, or methane if not.

- Combustion of organic material oxidizes carbon, producing among other things CO_2. Since burning fossil fuels releases carbon that has been stored in the Geosphere for millions of years it is considered on of the main reasons for increased atmospheric concentrations of CO_2.

- The solubility pump can also work in reverse order. At the surface of the oceans where the water becomes warmer, dissolved CO_2 is released back into the atmosphere.

- Volcanic eruptions release gases into the atmosphere. These gases CO_2.

- Reactions of limestone. Limestone, marble and chalk are mainly composed of calcium carbonate. As these are eroded by water, the calcium carbonate is broken down to form, among other things CO_2. Furthermore heating limestone, which produces a substantial amount of CO_2, is a common production process for cement.

(Above Janzen 2004, Berner 2003)

During times when carbon is taken from one pool to a greater degree than returned (e.g. photosynthesis exceedes respiration), we say that the pool is acting as a sink. The global carbon budget is the balance of the exchanges (incomes and losses) of carbon between the carbon pools or between one specific loop (e.g., atmosphere - biosphere) of the carbon cycle. (Chapin et al 2002)

An examination of the carbon budget of a pool or reservoir can provide information about whether the pool or reservoir is functioning as a source or sink for carbon dioxide. Some loops exchange more carbon than others, for example the amount of carbon taken up by photosynthesis and released back to the atmosphere by respiration each year is 1,000 times greater than the amount of carbon that moves through the geological cycle on an annual basis. (Chapin et al 2002, Cox et al 2000)

To measure this exchange the terms Net Ecosystem Exchange (NEE) and Net Primary Production (NPP) provide useful. NPP is simply the net carbon gain by vegetation and can be measured anually, i.e. it is the carbon bound in plants by photosynthesis minus the carbon lost in respiration. NEE on the other hand represents the net exchange of CO_2 between the ecosystems and the atmosphere. NEE can give a close estimate to Gros Primary Production (GPP or simply, net photosynthesis) and experimental approaches to measuring and determining environmental controls on NEE have been the focus of large research projects for many years. (Chapin et al 2002)

1.2 Ruminant physiology

A ruminant is a hoofed animal that ingests nutrition in two steps. The first step involves ingestion and semi-digestion of raw material into a substance known as cud. In the second step the cud is regurgitated and ingested once more, the entire process is referred to as ruminating. Ruminants include among others cows, goats, sheep, camels, giraffes, bison and deer. As such a large portion of the world's domestic farm animals are classified as ruminants. (Randall et al 1997)

From a modeling point of view most ruminants exhibit similar behavior in input and outputs. The specifics of the modeling approach used in this thesis will be covered in more detail in chapter 3 below. For now we will just point out the similarities between species of ruminants in the digestion and ingestion process. (Björnhag 2002)

All ruminants have a stomach with four chambers, the rumen, reticulum, omasum and abomasum. In the first two chambers the intake is mixed with saliva and separated into layers of solid and liquid material. Solids lump then together to form the cud which is regurgitated and mixed with saliva by chewing. The regurgitation and mixing in the first two chambers break down the intake into fibers, which in turn is broken down into glucose by symbiotic bacteria and protozoa. The symbiotic bacteria use almost all the glucose produced to support itself rather than its host. The ruminant receives a large part of their energy demands from volatile fatty acids produced by the bacteria. As such it is

interesting to note that ruminants do not live directly of the food they ingest but rather ingest food to feed bacteria whom in turn feed the ruminants. The broken-down fiber then passes through the rumen into the orgasum, where water is removed. As a final stage the food is moved into the aborgasum where the broken-down fibers are ingested much like it would be in a human stomach. Finally the food is sent to the intestine where the absorbtion of nutrients can take place. (Björnhag 2000, Randall et al 1997)

2.2 Ecological and biological modeling

Modeling is an essential and inseparable part of all scientific, and in actual fact all intellectual, activity. Often when scientists explore a system or process they utilize an algorithm. An algorithm or mathematical model is the use of mathematical language coupled with logic to describe the behavior of a system. Algorithms are commonly used in both natural sciences (such as physics, biology, and electrical engineering) and social sciences (such as economics, sociology and political science). An algorithm usually describes a system by a set of variables and a set of equations that establish relationships between the variables.

Modeling can be argued to be as much science as an art form; this is especially true for modeling of biological systems. Scientific modeling is a vast subject and a lot of effort has gone into explaining and categorizing the process, in fact many authors have devoted entire careers to the subject. I will try to avoid expanding too much on the subject below, but I feel it is unavoidable to discuss some of the theoretical problems associated with the aim of this paper.

2.2.1 Setting the complexity of the model

A basic issue when constructing a model is at what level of complexity the model shall function. If one were, for example, modeling the launch of a space rocket, one could insert each mechanical part of the rocket into a model and couple it with each physical law that affects the launch. However, this approach has several drawbacks. For example the computational cost of adding such an enormous amount of detail could render the model inefficient. Furthermore, the uncertainty inherent to an overly complex system would increase, since each part generates some amount of variance into the model. It is therefore always appropriate to make some approximations to reduce the model to a sensible size.

Further more one can question whether it is even possible to define the complexity of a system independently of an observational context. For example, consider an apple falling in a uniform gravitational field, basic textbooks in the subject of physics present this as one of the simplest systems imaginable. However, this “system” is actually a model of a real system. The physicist’s view of the system would differ from the viewpoint of say a botanist studying the same system. Even if we convince the researchers that the size, ecological function and colour of the apple are not really part of the system, we cannot be

completely certain ignoring the small but real effects associated with for example radiation pressure may be appropriate. Any real system is infinite in its complexity, an important task for the modeler is applying a correct level of complexity in a sober manor with regards to the observers level of sophistication, thus avoid the inclusion of material that will make the development of the model more difficult without making its performance any better. The example used above can for most purposes be successfully modeled by a very simple model first described by Isaac Newton. (Klir 1993)

Some might object to the line of reasoning for infinite complexity by arguing that insofar as systems possess fundamental complexity, all systems possess equal complexity. The basis for this claim is that virtually all systems are embedded in a hierarchy of linked subsystems, and we tend to define each of these subsystems in the context of what we, as observers, find interesting enough to investigate and are able to measure. (Klir 1993)

Regardless if one is to introduce complexity as a meaningful concept in the process of modeling, then it must not be considered an intrinsic property of systems. Rather it is a property of models of systems. Newton's model of a falling apple is not simple because a falling apple is simple. It is simple because Newton used a model that considers only the motion of the centre of gravity and ignores all aerodynamic and biological effects. (Klir 1993)

The usual approach to model development is to characterize the system, make some assumptions about how it works, translate these assumptions into equations, and start summarizing it into an algorithm. This does not always work. If the model does not fit the evaluation data, it may be the data that is inaccurate or it may be that the algorithm (i.e. assumptions) are wrong.

If one has no reason do believe ones data is inaccurate an alternative approach often called “top-down modeling” or “system identification” can be used. This method works backwards when compared to the traditional and asks what kinds of models could fit the data rather than making a set of assumptions about the system and creating models on the basis of these assumptions. The approach has the integrated benefit of guaranteeing the availability of suitable data.

In any discussion of systems complexity, biological systems stand out as paradoxical. Biological systems are often without equal in complexity, and yet the most widely used biological models are extremely simple. Part of the explanation for this paradox is the connectivity biological systems exhibit, as explained earlier all systems can be considered parents of sub systems and when it comes to biological systems this is especially true and certain sub-systems can not easily be ignored. In some cases this leads to the holistic approach being the only satisfactory in order to explain data without creating overly complex models. Another part of the explanation is the data itself, typically data can be difficult or impossible to acquire at an appropriate detail and if high detail is acquired the data can be tainted with high amounts of noise. Thus increasing the uncertainty in the results. (Bjørnstad and Grenfell 2001, Maurer 1998, Levin et al. 1997)

2.2.2 Model calibration

Any model of complex systems contains parameters or variables that are difficult to measure or estimates of their true states. In some cases these variables can be adjusted within their confidence intervals until the model fits the expected results. Indeed, as many parameters in ecology have large confidence intervals this approach is relativly common practice.

2.2.3 Model evaluation

An important part of the modeling process is the assessment of model performance. How can we find out if the mathematical model describes the system well? This is often not an easy question to answer. A common approach is to split the measured data into two parts; *training data* and *verification data*. The training data is used to calibrate the model (see above). Whereas the verification data is used to evaluate the models accuracy. Assuming that the training data and verification data are not the same, and if the model describes the verification data well, then we can often assume the model describes the real system well.

However, this still leaves the question if this model describes events outside the measured data satisfactory. At this point it is important to consider that the model is simply a model of the particular system and not a simulation of reality. As such, attempts to use the model to describe situations it was not specifically designed for must be taken with caution and assumptions need to be re-evaluated.

2.2.4 Carbon cycling modeling

Modeling the earth's carbon cycle is an exercise that at any rate must find ways to traverse all the above-mentioned situations, and often many more. This is not a trivial exercise and developing a model can take years, this leads to the unavoidable result that there are a diverse range of solutions and approaches. One approach that has gained support recently is the modular process-based model. The process-based approach divides the carbon cycle into processes at various levels of detail that feed each other with in- and out-puts. It groups certain mechanisms into functional types and treats them to similar rules. This allows for an extensive flexibility, the process-based approach can calculate details which we know a lot about such as leaf-level photosynthesis at a different time-scale and ecological level as something we are hold at greater uncertainty, for example, soil organic matter decomposition. (Janzen 2004, Berner 2003, Chapin et al 2002 Hastings 2000, Yodzis 1989)

One incentive for carbon cycle modeling becomes clear when models of the carbon cycle are incorporated into global climate models, so that the interactive responses of the oceans and biosphere on future CO_2 levels can be modeled. As rising atmospheric CO_2 concentration and its potential impact on future climate are an issue of global economic and political significance the need to understand how carbon cycles through the Earth-

system becomes important to our ability to predict any future global change. (Alley 2003, IPCC 2001)

3 Methods and Data

3.1 History Database of the global Environment geographical data

The History Database of the global Environment (HYDE) (Klein Goldewijk, 2001) version 2.0 offers global land use data at the same scale as the LPJ-DVGM model operates at (0.5º by 0.5º longitude/latitude grid). HYDE was originally intended for testing and validation of Integrated Model of the Greenhouse Effect (the IMAGE model) and employs state-of-the-art models based on data acquired from the Food and Agricultural Organization of the United Nations (FAO) to classify land use.

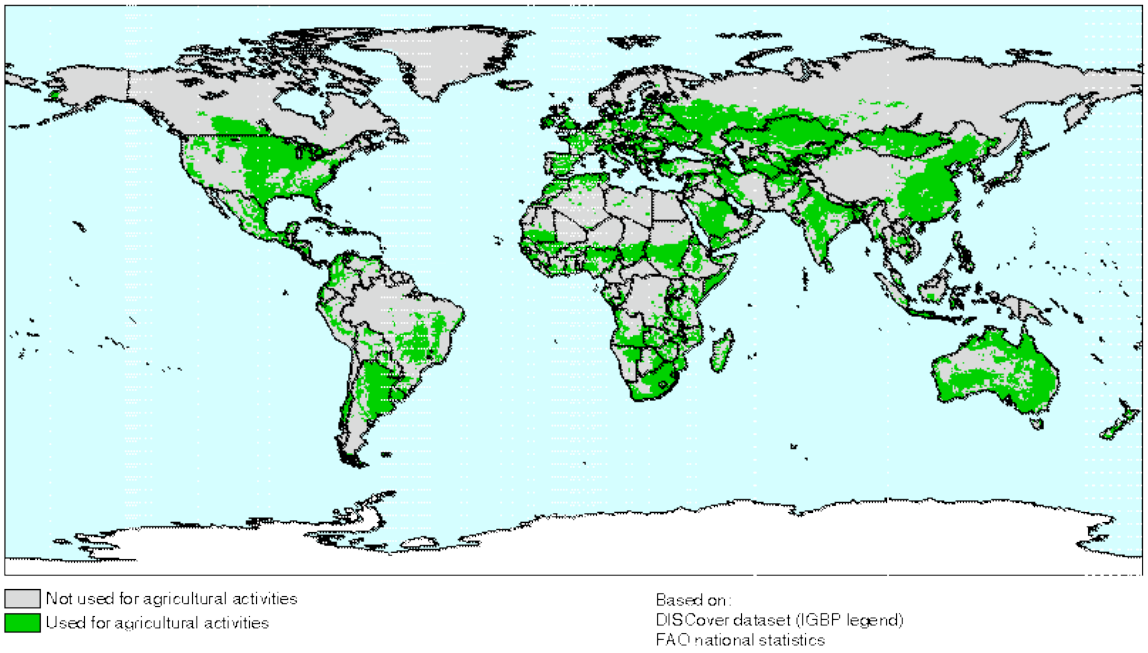

Figure 2 Input for land use calculations for the initial creation of HYDE 2.0, based on DISCover dataset and FAO national statistics in Goldewijk 2001

Allocation of cropland and pastures in the HYDE database were guided by the following rule sets;

First the allocation of cropland was computed using agricultural land cover derived from the DISCover land use dataset, which was reclassified according to the International Geosphere Biosphere Programme Global Land Cover Legend (Loveland and Belward, 1997). This scheme consists of 17 classes, of which “Croplands” and “Urban and built-up” were used for allocation of the amount of “Arable Land and Permanent Crops” given

in FAO's national statistics. If the requirement for arable area could not be satisfied with these two categories, the classes "Savannas, Grasslands and Cropland/Natural Vegetation Mosaic" were used to allocate the rest.

Secondly, the total amount of cropland was allocated according to historical population density maps on a sub-national or country scale. The grid cells with the highest population densities were initially assigned to cropland, then those with the second highest density, and so on, until the total amount of cropland was allocated in that element.

Thirdly, the allocation of cropland was restricted to the agricultural area as determined by the initial land cover map, which is considered representative for present agricultural activity.

Lastly, the total amount of pasture in a nation was allocated according to population density in a similar way as for croplands, excluding those grid cells that were already allocated to cropland on the basis of the above-described rules. This process results in figure 4 seen below.

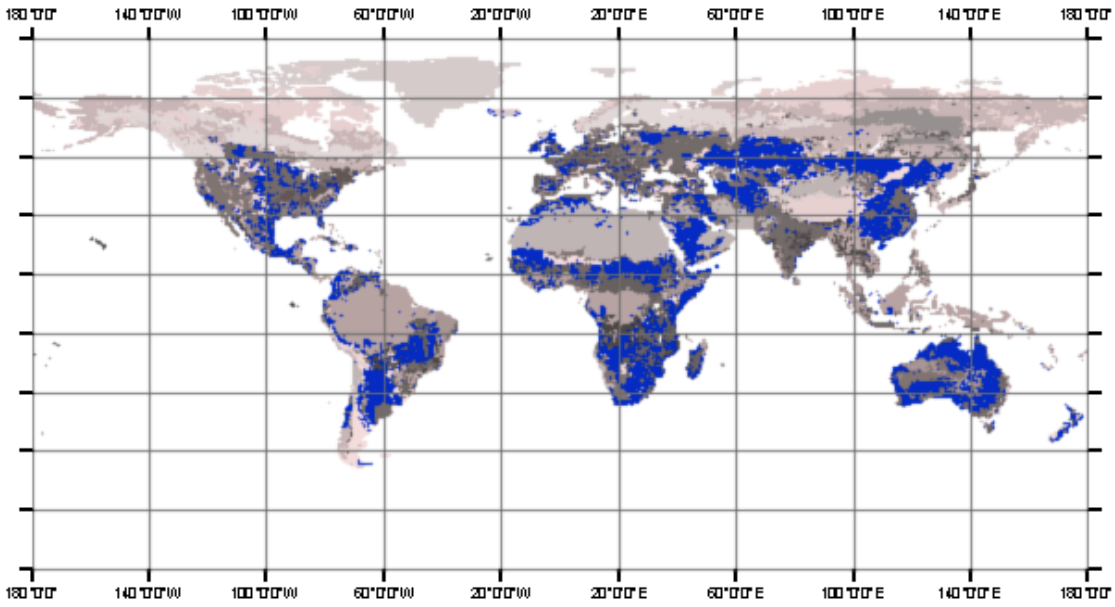

Figure 3 Cover of the land use type; pasture in blue colour - as calculated by HYDE 2.0 using data for 1990

There can be many uncertainties identified in the HYDE approach.

- Sometimes data are only available in local libraries, or through researchers, and are not known to the research community. This leads the drawback of not having any other data sources to verify or compare against.
- Data are often only available in hard copy, not in digital format. Often the digitizing from hard copy to digital is prone to errors.

- The data source itself has sometimes been known to be inaccurate through manipulation by governments, guestimates, or poor measurement methods.
- Allocations of areas of cropland and pasture have been coupled to population densities. It is questionable if population density is an acceptable proxy for the distribution of cropland and pasture. It seems reasonable to assume so but it is far from certain.

Goldjewijk has in his work on the HYDE database applied the following solutions in order to alleviate some of the problems mentioned above:

- For modeling purposes and analysis it is unavoidable from time to time to manipulate the original data in order to get reasonable figures. Interpolation techniques have been applied sometimes to fill in gaps.
- Regional rates have been applied to separate countries in that region, in order to get an estimate of error.

However, even though there is no real satisfying solution for the problems mentioned above these problems are not unique to the HYDE database. Rather any attempt to map global patterns will suffer the same problems.

3.2 Animal population data (FAOSTAT)

Population data for herbivores was collected from FAOSTATs website (FAO 2005 (last updated February 2005)) on the 30thy of July 2005. Population data was aggregated across the six continents North America, South America, Europe, Africa, Australia and Asia (a breakdown per country for each continent can be found at FAO's website).

As the model already uses static geographic data for the pastures the same constraint was applied to the population's data, therefore one year's data had to be chosen to represent the modeling period. There are several shortcomings to this approach, among which, is the fact that global rudiment populations have increased tremendously over the modeling period (1970-2000 has seen a 30% increase in rudiments) and thus it the approach taken adds uncertainty. However, to what extend has not been investigated. The year 1996 was chosen for the modeling chiefly for two reasons;

1. Under the period covered by the climatic data 1996 was the year with the highest populations of rudiments.
2. Reliable data for methane emissions exists for 1994. As this data will serve as a control for the model a year as close to this and present day as possible will ensure best possible control data.

By choosing population numbers at the high end of the spectrum a short coming is introduced that could have been alleviated by choosing population numbers closer to a median for the period. The down side will be that the further from 1996 we go in time, the more uncertainties we will get in the results. Likely to compound this is the additional

stress the model will be under during dry years. However, if nothing else at least 1996 will provide some interesting data for analysis.

The data set used can be seen in Table 1 below.

Stocks (Head)	North America	South America	Europe
Cattle	166,954,740	289,656,298	172,420,343
Goats	14,287,841	17,635,899	19,088,708
Sheep	17,577,532	79,851,371	168,224,520
Total per Area	198,820,113	387,143,568	359,733,571

Stocks (Head)	Africa	Australia	Asia
Cattle	203,041,920	26,377,400	449,773,248
Goats	195,527,275	230,000	441,843,820
Sheep	214,188,775	121,116,064	411,936,117
Total per Area	612,757,970	147,723,464	1,303,553,185

	Total per species
Cattle	1,308,223,949
Goats	688,613,543
Sheep	1,904,511,447

Table 1 Global herbivore population data used for the grazing subroutine (FAO 2005)

The data suffers similar problems as the land use data mentioned above. Without going into details the measures FAO have used to combat this are also generally the same as those used for the HYDE database. Notwithstanding, as FAO statistics are generally accepted as credible and are widely cited in journals they can be considered sufficient for the purpose of this thesis.

3.3 Data processing

On the onset of the thesis about half of the time allocated towards completion was dedicated for data processing, even so this part of the thesis took much longer than its generously allocated budget allowed. It is estimated around 70% of the initial time budget was spent data processing. As mentioned in the preface the thesis was initially thought to be achievable in 400 hours of work. However, once this time had been spent it was clear given the scope of the thesis the amount of work necessary for completion had been underestimated; and some supplementary time in the coming weeks was devoted to

additional data processing. In total the time spent for the thesis has twice exceeded the initially intended, and data processing has constituted at least 300 hours of work.

In this section I will briefly explain the data processing process and some of the complications experienced. As such this section is aimed at the reader wanting to pursue similar endeavors, any other reader can safely skip this section without loosing any context.

One of the main reasons why data processing has been a significant time sink is because it involved developing specialized software to process and verify the integrity of pre- and post-processed data. The specialized software was necessary since the data collected from various sources was generally of incompatible formats; this was further compounded by the fact that the specific data formats often were poorly documented. It is important to remember that there are many ways to skin a cat and the techniques described in this section neither claim to be the best available nor did I extensively test different methods to find the most efficient, it is simply what worked for me and how I did it. Part of it I would without doubt do in a different manor today, but as some data processing was necessary prior to building the model and working with LPJ-DGVM it constituted some of my first experiences with C++. (for some archetypical data processing source code see Appendix 2 (1))

For the sake of presentation the data processing process is divided into pre- and post-processing, where pre is anything necessary before the LPJ-DGVM model could run with the grazing subroutine and post is anything done with the models output.

3.3.1 Pre processing

In order for data to be used as input for the LPJ-DGVM model it needs to be organized in a certain order. With the intention to avoid introducing as much new code into the LPJ-DGVM source as possible it was decided to reformat all new input data into something that will resemble input data already used in the LPJ-DGVM model. It was decided to mimic the data structure LPJ-DGVM already utilizes as an input in "cru_world_lon-lat.txt". Thus a series of small stand-alone programs were created. This approach had several advantages, chiefly optimization of LPJ-DGVM with grazing subroutine as the runtime stayed the same with no unnecessary time-consuming loops, but also debugging several small programs proves easier than debugging one large.

After several failed attempts at formatting the input data into the desired format and unsuccessful attempts to research how the data file Goldewijk (2001) provides was organized, a visualization application was written. The visualization application utilizes OpenGL to read space-delimitated text files and draws 0.1 times 0.1 boxes without depth in one plane (open source window management code provided by Jeff Molofee 2000). The visualization application alleviated the file format problems encountered in the Goldewijk (2001) files and the data was successfully reorganized according to fit the

aforementioned criteria. The data was also coded for continental association as displayed in figure 6 below.

The visualization application was later recycled for various data processing verification tasks as it proved much faster to rewrite the visualization application than to import files into ArcMap for display. One variation of the visualization application can be found in Appendix 2 (2) and three screenshots are shown below.

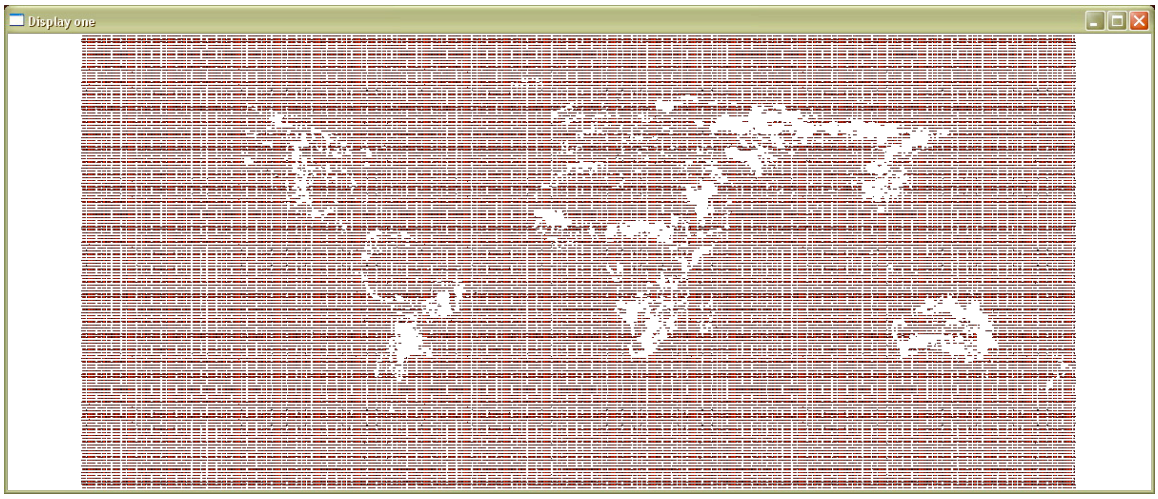

Figure 4 Illustrates the basic operation of the visualization application. Goldewijks (2001) is displayed with areas used for grazing in LPJ-DVGM negatively coloured.

Figure 5 Illustrates a typical pre processing failure, with grid cells for various continents misplaced.

Figure 6 Illustrates the first successful attempt at organizing Goldewijks (2001) data in the correct order. Continental borders used for the gazing subroutine are illustrated with different colours.

After this some time was spent processing the output data of the model in order for it to take a shape suitable for presentation in this thesis.

3.3.2 Post processing

Data yielded from running the LPJ-DVGM model over the whole earth with the grazing subroutine turned on exceeds two GigaBytes and is thus unmanageable without some post processing. Post processing involved three basic steps.

Since the LPJ-DVGM model assumes all grid cells are of equal area, when in fact a grid cells size is dependant on geographic location, a program to adjust for this had to be written. The program would recalculate data for each grid cell according to a formula for the surface area of a segment of a hemisphere of radius r from the equator to a parallel (circular) plane h vertical units towards the pole:

$$S=2*\pi*r*h.$$

Thereafter the data would be reformatted into space delimitated text files similar to Goldewijks (2001) format as this was found to be a suitable import format for ArcMaps ASCII to raster function. Therefore yet another program was written from recycling

various preprocessing programs, as it was essentially the same process but backwards. An example can be seen in appendix 2 (3).

Finally the ESRI ArcInfo software suit was used for import and layout of the data. The results of the post process can be seen throughout the thesis and for example in figure 3 above (section 3.1).

3.4 LPJ-DGVM ecosystem model

LPJ-DGVM (Smith et al. 2001, Sitch et al. 2003, updated hydrology by Gerten, D. et al. 2004) is a dynamic ecosystem model that simulates ecosystem processes and vegetation distributions and dynamics. See process based approach under carbon cycle modeling above (section 2.2.4). LPJ-DGVM simulates the global carbon cycle through a well-established and broadly validated ecosystem model.

The model is written in standard C++ and is modular in its code. This makes it highly customizable and simple to add functionality too. Indeed this was a design criterion for its first version.

For a more complete analysis of LPJ-DGVM's approach towards carbon cycle modeling please refer to Smith et al. 2001 and Sitch et al. 2003.

3.5 Model implementation of grazing subroutine

A sub-routine was added to the LPJ-DGVM model in order to integrate herbivore grazing. The subroutine is for the most part contained in the growth.cpp source file written by Ben Smith(2002) this and all changes to the LPJ-DGVM source are detailed in appendix 1. Below I will describe the main functions of the grazing subroutine and the manor it was implemented.

The model design was to some degree "top down" (see section 2.2.4 above) as the available data favored this approach.

The first step was to create a causal loop diagram (figure 7) in order to map out the plausible relationship of the data available. An important step was to make a central assumption. Traditionally in ecology, plant-herbivore interactions have been considered to have a negative effect on plants. This assumption has been under debate for some years. Some researchers suggest that herbivores can have a positive effect on plants and their productivity. (Leriche et al. 2000, Paige and Whitham 1987) For the purpose of this paper it is assumed that herbivore grazing reduces grass biomass. Further population density is assumed to be the driving force behind the model. (Nowak and Sigmund 2004) It is also assumed the herbivore population is homogenous in the sense that any individual is the same as the next. Thereafter some mathematical equations were researched from literature and implemented into the model. The subroutine is calculated

using the total mass of all herbivores present in the current grid. It determines the amount of vegetation needed for maintenance that is subtracted from the grasses biomass.

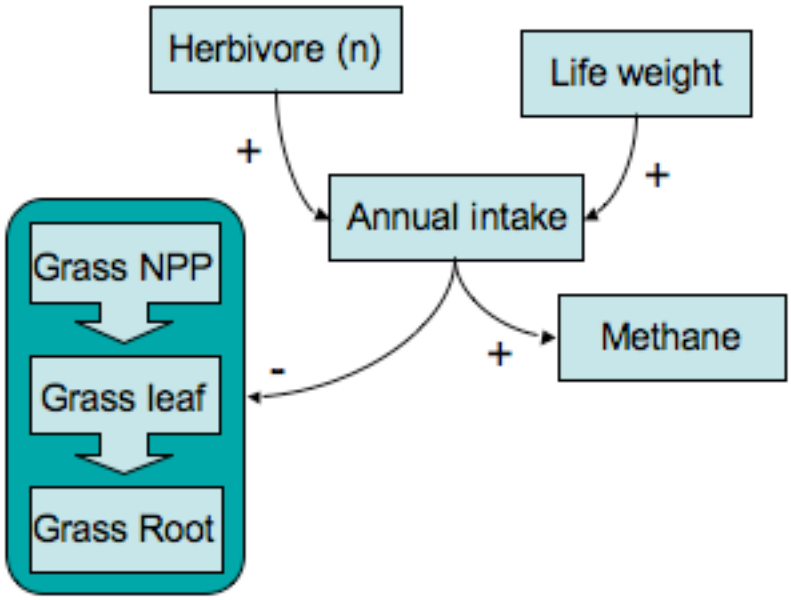

Figure 7 Original causal loop diagram of the grazing subroutine's operational flow

Maintenance costs and possible biomass intake rates of herbivores are linearly related to their metabolic weight. Metabolic weight is an exponential function of life weight; as such larger herbivores have exponentially lower maintenance costs per unit of weight than smaller counterparts. These relationships are well documented and further explained by for example Björnhag (2002, 2000) in his compendiums "Små djur och stora" and "Växtätarna".

$$m = l^{c}$$

Where m is metabolic mass, l is herbivore life weight and c is a constant for metabolic mass from life weight set at 0.75.

In the model herbivore life weight is set at 250 kg (as an approximate mean between the life weights of cattle, goats and sheep and factored for their portion of the global totals) and metabolic mass from life weight is set at 0.75. (Parsons et al. 2001, Kleiber 1967)

Density of herbivores in the grid is calculated through herbivores population size, n, divided by area of grid multiplied by metabolic mass.

$$dens = n / (area*1000000)$$
$$m_{m2} = m * dens$$

Possible daily intake of plant carbon is calculated from possible intake parameter, a, multiplied by plant carbon fraction, p, multiplied by metabolic mass per m^2.

$$I = a^* p * m_{m2}$$

In this calculation a is set at 0.06, and p is set at 0.45. Where a is a training variable and can be set between 0.04-0.08. (Karin Nadrowski personal communication 2005)

Only a fraction of the ingested vegetation can be used for maintenance (met). Some of it is lost to urine, faeces and methane. The subroutine calculates flux from methane and flux added to biomass from ruminants by the following equations;

$$ufc = aIc^* v_q * u_c$$

$$met = aIc^* v_q * (1.0 - u_c - m_f)$$

$$\text{annual carbon flux for ruminants} = met + ufc$$

$$\text{annual methane flux} = aIc * v_q * m_f$$

Where aIc is annual intake of carbon, v_q or vegetation quality is a constant set at 0.6, u_c is a constant for urine fraction set at 0.04, and m_f fraction of methane is set at 0.08. (Karin Nadrowski personal communication 2005)

The last step enables the model to be evaluated by calculating herbivore methane gas production.

On a yearly basis the subroutine subtracts herbivores intake (aIc) from the grassland biomass in one of three ways (figure 8 below). The ingested biomass is first subtracted from the annual NPP. However, if the annual NPP is smaller than annual intake, leaf and root biomass are subtracted the remainder. In the last case, if the ingested biomass exceeds all three possible sources (NPP, the root and leaf component of grasslands biomass) the annual intake is set to "annual NPP + 0.9 * the root and leaf component of grasslands biomass" and then subtracted accordingly from each component. This action also triggers a warning message to be printed to a log file.

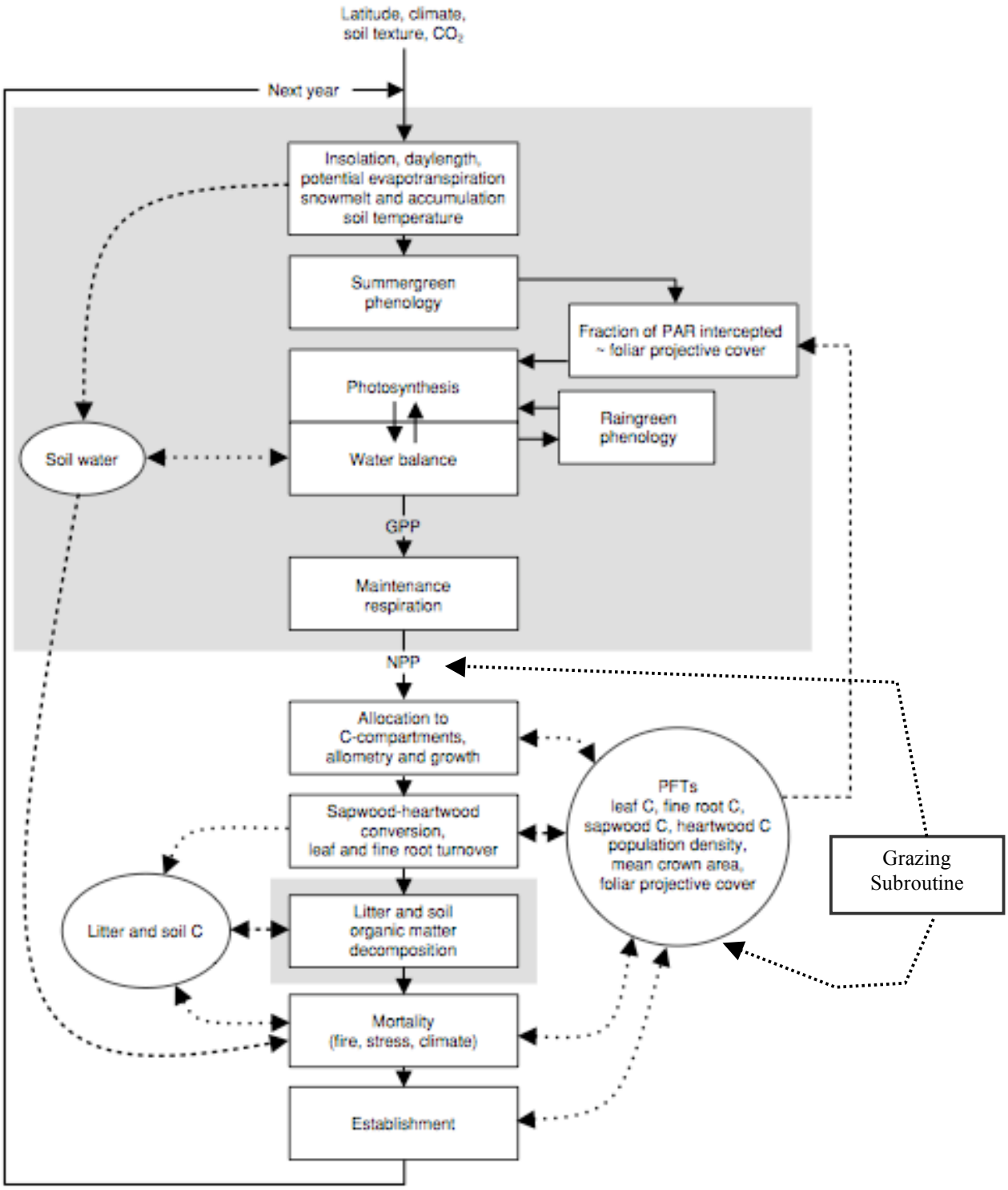

Figure 8 A flow chart describing individual processes in the LPJ-DVGM model. Boxes with shaded background are called monthly or daily, and boxes with white background (lower part of figure) are called annually. From Sitch et al 2003 modified to show the grazing subroutine.

3.6 Modeling protocol

The LPJ-DGVM model was run twice; once with the grazing subroutine turned on and once with it turned off. Thereafter the grazing implementation (grazing subroutine turned on) is compared to simulated potential natural vegetation (grazing subroutine turned off). Both runs were allowed a standard 300 year spin up period after which forests were cut down without taking into account the carbon release from this action. Thereafter the subroutine grazing was activated at year 368.

In the results section below (section 4), the output data for the last 30 years covered by climate data (corresponding to 1970-1998) has been aggregated.

4. Results and Discussion

The results for NPP are straightforward and support the thesis statement that herbivore grazing has a significant impact on the global carbon cycle. Indeed figures 10a and 10b show a lower productivity over the studied period of about 5%, which follows the same trend as other studies such as Leriche (2001). However, it is difficult to find literature that agrees on a precise amount of decrease to expect as this relies on many factors not incorporated into the model. If such a number could be agreed upon the model could be trained to match this. At the current state it is acceptable that the model shows the correct trend.

The results for NEE correspond to the interannual patterns of variability predicted by Sitch et al. (2003), despite this the results (illustrated by figures 9a and 9b) are very surprising and difficult to account for. The model seems to suggest the biosphere is acting as a 50% stronger sink if subroutine grazing is activated. An explanation could perhaps be sought in that the model inputs larger quantities of litter into the soil of grasslands than it does for forests. Thus grazed grasslands would have higher carbon storage than the woody vegetation types. One possible mechanism for this theory is that currently the LPJ-DVGM does not calculate fires in grasslands correctly (Thomas Hickler personal communication 2006). However, it is important to be cautions at this point and more research is required in order to explore this statement further.

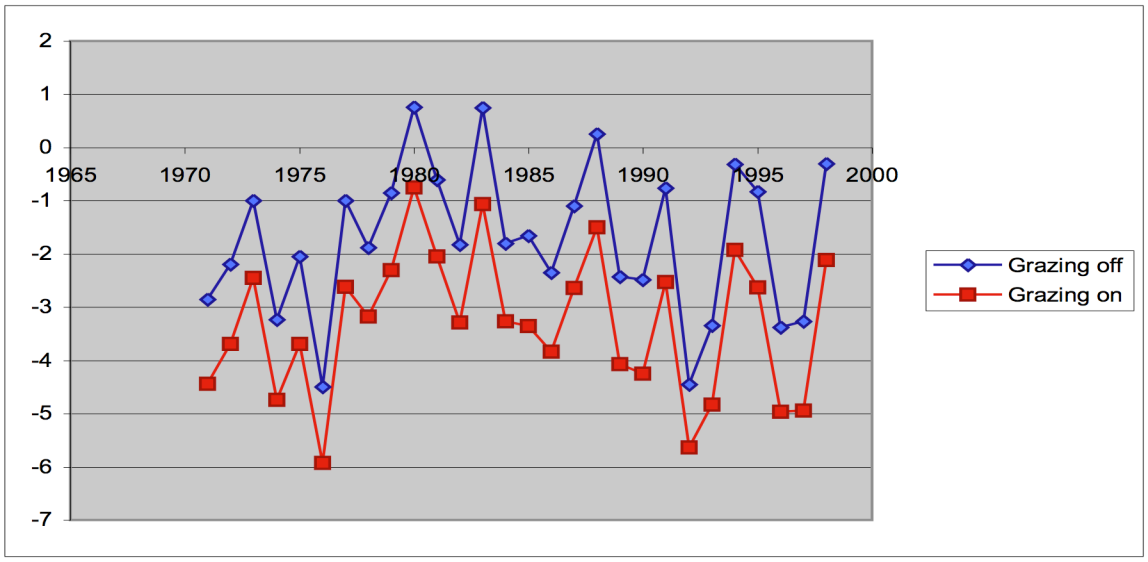

Figure 9a Annual global NEE with the grazing subroutine on and off (P ton C / year)

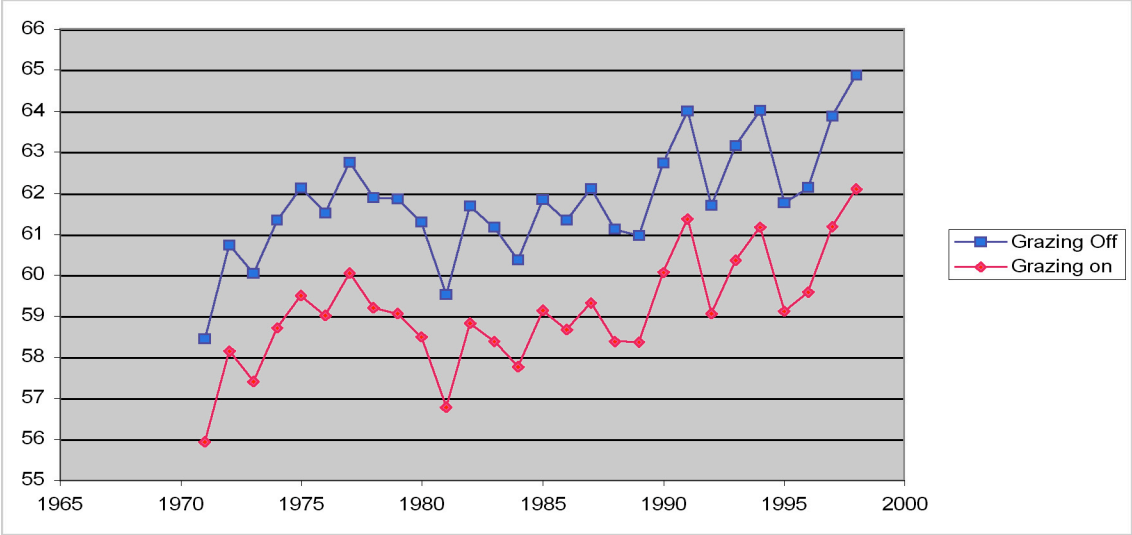

Figure 10a Annual global NPP with the grazing subroutine on and off (P ton C /year)

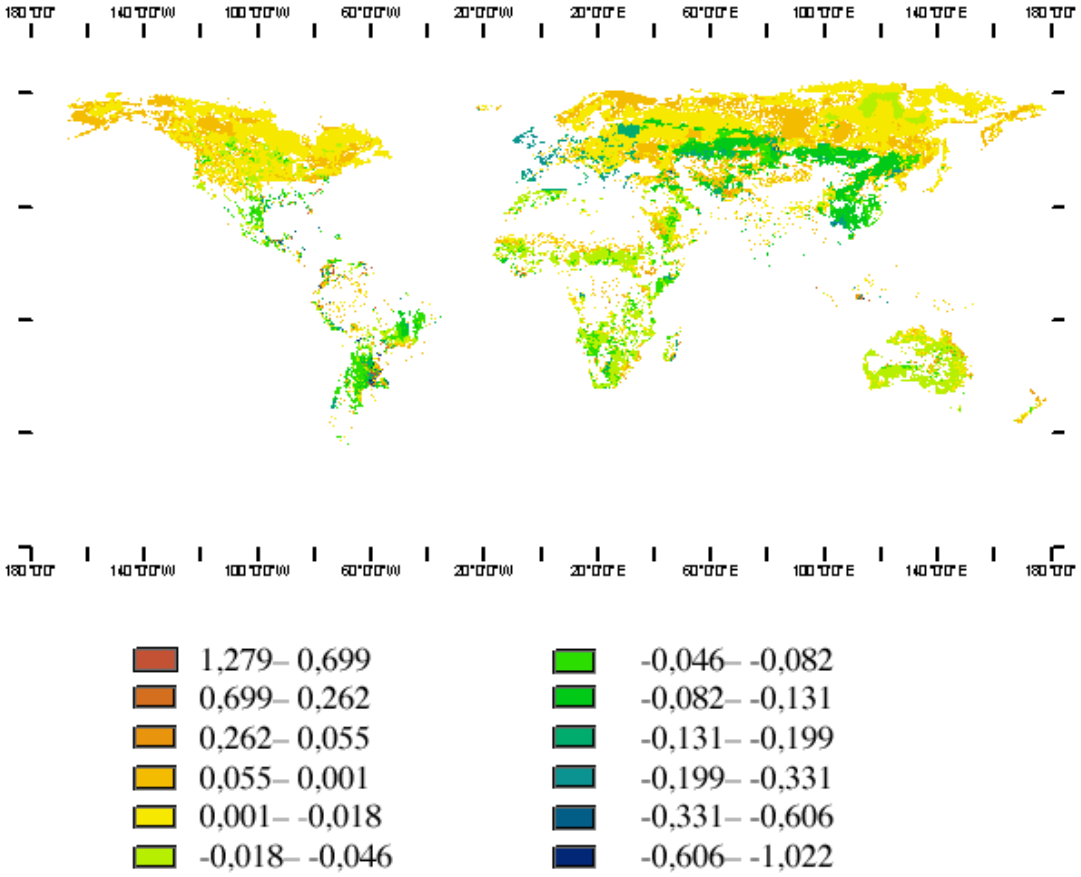

Figure 9b Difference in NEE between the grazing subroutine turned on and off for each cell in the LPJ-DGVM model. (g C / m^{-2})

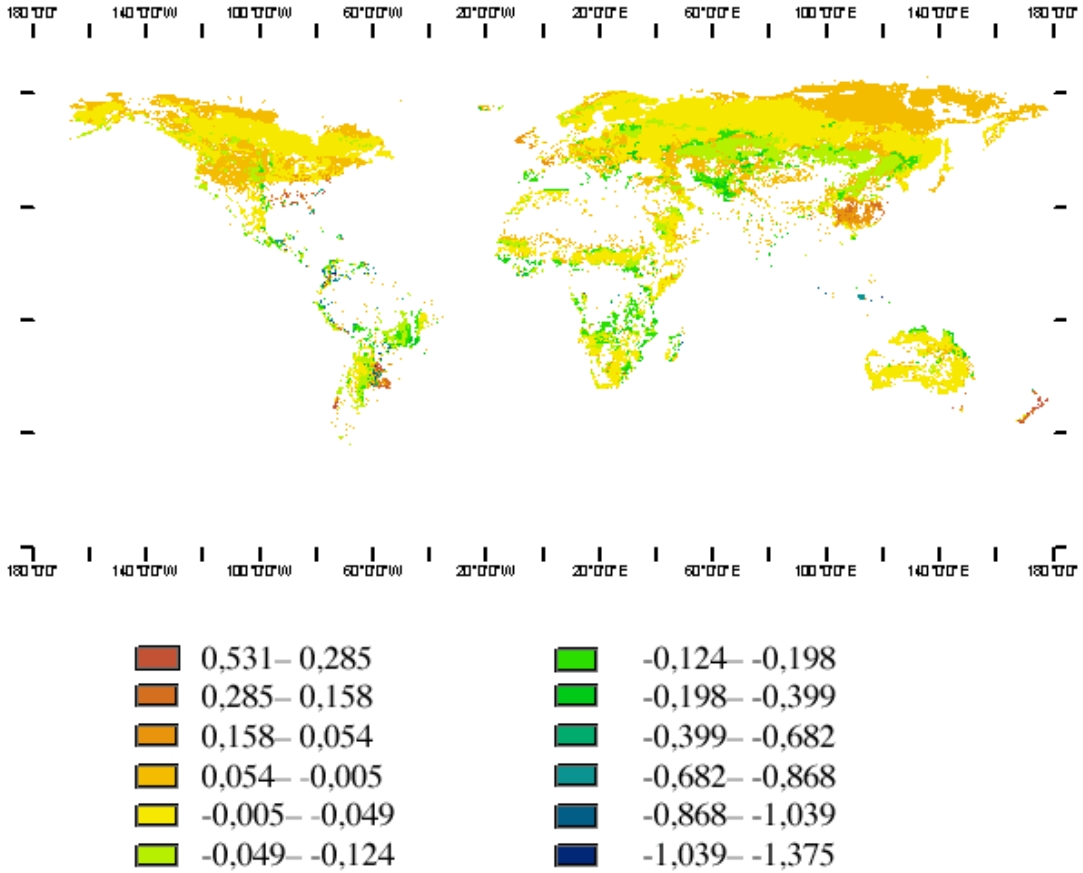

Figure 10b Difference in NPP for the last model year (497) between the grazing subroutine turned on and off for each cell in the LPJ-DGVM model. (g C / m^{-2})

Further analysis showed that the spatial distribution of the changes in NPP and NEE are distributed fairly centrally over the areas marked out for grazing by the model (figure 3, section 3.1). There is a notable spillover of change outside of expected areas of change and no suitable explanation for this can be found at present. It is assumed that the post processing software has a yet unaccounted for bug that can explain the spillover effects. However, in order to merit this claim further examination is called for.

All in all contradictory results are common in this line of research, especially in the early stages, and a search for the mistake/mistakes can take months before it yields a tenable result.

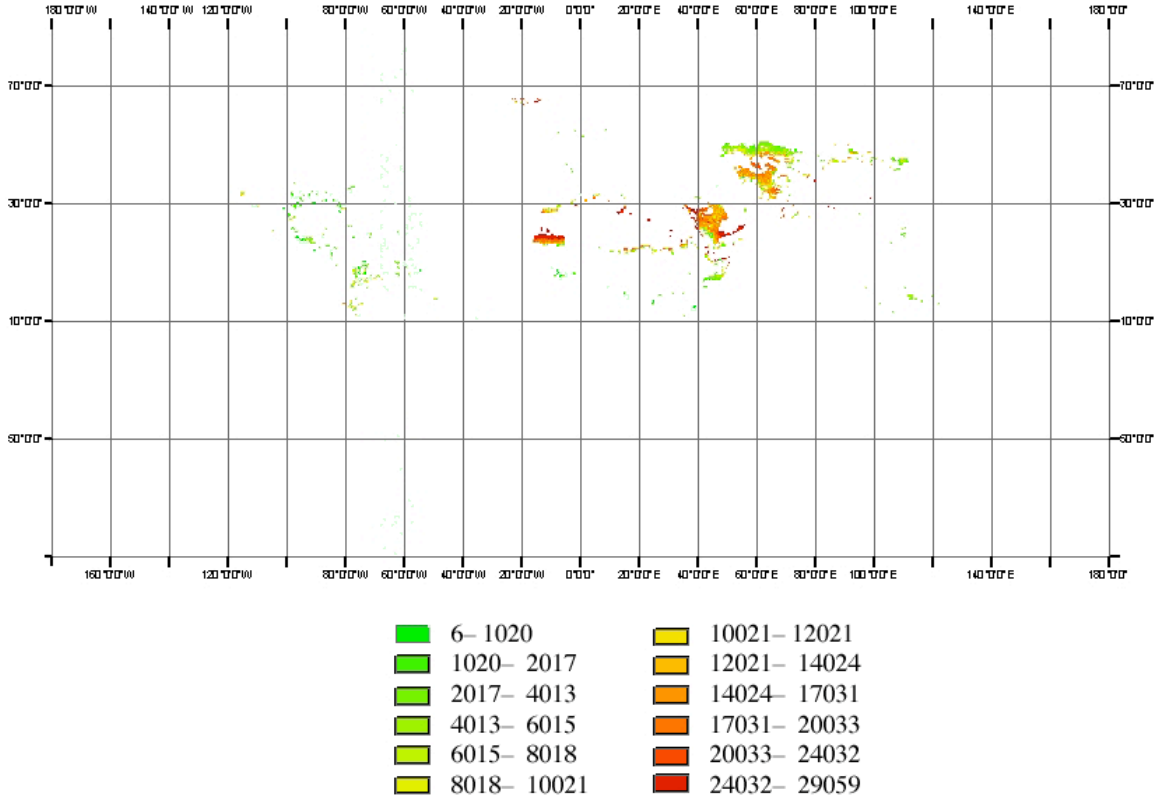

Figure 11 Is a quantitative representation of amount of errors in the grazing subroutine (see above section 3.5), i.e. when the grazing function required more input than the ecosystem could produce, over the entire study period (1970-1996). The results are aggregated in the figure by adding one to the grid cell for each error and a thousand for each error in a year directly following a previous error. Thus 9012 in the figure represents 12 years of error in the study period, 9 of which were adjacent to a previous error.

The majority of errors are mainly located in the dryer areas of the world. This provides confidence, as it is an expected result of the homogenous herbivore populations. As the grazing subroutine grazes the less productive lands in the same manor as the more productive a large amount of errors was expected in the less productive. The error distribution is illustrated by figure 11 above, where North Africa and the Middle East exhibit a larger amount of error than the rest of the world.

USEPA (1994) estimates total global methane emissions for 1994 at about 78.5 million tons. The model output for the same year is 88.6 million tons. The methane emission of livestock is a good test of the model, and it is interesting that the model estimates are so close to other independent ones.

The model and its interpretation is a first step and much can still be learned from its data. As such areas of future investigation are boundless and not only limited to development

of the model but also towards more detailed data mining of the accumulated results. To exemplify a few areas of model improvement these can include, but are not limited too;

- Livestock populations are rarely homogenous and static populations. Rather, as commented earlier, they often increase and decrease in size and weight and thus grazing pressure. The model uses a rather crude spatial distribution of livestock numbers, only a continental scale was considered, and an undynamic population. As such a more complex grazing pressure could be modeled, both spatially and over time.

- No calculations were made to account for the carbon cycle effects of the land use changes livestock practice demands. This is an important omission. As mentioned earlier carbon is released by forest clearance to transform forests into pastures.

- Other types of verification data could be calculated, such as population weight gain (or production) and compared against global and regional meat production. This process could add more validity to the model if the data output is within an acceptable range.

- The population dynamics between predator (ruminant) and prey (grasses) are very simplistic and research into the value of expanding on this could be worthwhile. Since the dynamics can in some places be quite complex and add, for example, feedbacks when the prey is overexploited. However, since ruminants, as defined by the thesis, are by large managed populations it was deemed not worthwhile to pursue within the scope of this thesis.

5. References

Alley R.B. Marotzke J, Nordhaus W.D., Overpeck J. T., Peteet D. M., Pielke Jr R. Pierrehumbert R. T., Rhines P. B., Stocker T. F., Talley L. D. and Wallace J. M. 2003 *Abrupt Climate Change* Science 299, 5615, (2005 – 2010)

Berner R.A. 2003 *The long-term carbon cycle, fossil fuels and atmospheric composition* Nature 426, (323-326)

Bjørnstad O N. and Grenfell B T. 2001 *Noisy Clockwork: Time Series Analysis of Population Fluctuations in Animals* Science 293. 5530, (638 – 643)

Chapin III F. S., Matson P. A. and Mooney H. A. 2002 *Principles of terrestrial ecosystem ecology* Springer

Cox P.M., Betts R.A., Jones C.D., Spall S.A. and Totterdell I.J. 2000 *Acceleration of global warming due to carbon-cycle feedbacks in a coupled climate model* Nature 408, (184 – 187)

Gerten, D., S. Schabhoff, U. Haberlandt, W. Lucht, and S. Sitch. 2004. *Terrestrial vegetation and water balance - hydrological evaluation of a dynamic global vegetation model* Journal of Hydrology 286, (249-270)

Hastings A. 2000 *The Lion and the Lamb Find Closure* Science 290. 5497, (1712 – 1713)

Thomas Hickler 2006 *personal communication* Lund University

IPCC (Intergovernmental Panel on Climate Change), *Climate Change 2001: The Science of Climate Change. Report of Working Group I* (Cambridge Univ. Press, Cambridge, UK, 2001)

Janzen, H. H. 2004 *Carbon cycling in earth systems—a soil science perspective* Agriculture, ecosystems and environment 104, (399 – 417)

Kleiber. M. 1967 *Der Energihaushalt von Mensch und Haustier* Paul parey

Klein Goldewijk, K. 2001 *Estimating global land use change over the past 300 years: the HYDE database* Global Biogeochemical Cycles, 15(2) (417-434)

Klir, G. J. 1993 *Systems science: a guided tour* Journal of Biological Systems 1 (27-58)

Leriche H., LeRoux X., Gignoux J., Tuzet A., Fritz H., Abbadie L. and Loreau M. 2001 *Which functional processes control the short-term effect of grazing on net primary production in grasslands?* Oecologia 129, 1, (114–124)

Levin S A., Grenfell B, Hastings A and Perelson A S 1997 *Mathematical and Computational Challenges in Population Biology and Ecosystems Science* Science 275. 5298, (334 – 343)

Loveland, T.R. and Belward, A.S., 1997 *The IGBP-DIS global 1km land cover data set, DISCover: first results* International Journal of Remote Sensing, 18. 15, (3289-3295)

Karin Nadrowski 2005 *personal communication* facilitated by Thomas Hickler of Lund university in August 2005

Nowak M.A. and Sigmund K., 2004 *Evolutionary Dynamics of Biological Games* Science 303. 5659, (793 - 799)

Maurer B.A. 1998 *Ecological Science and Statistical Paradigms: At the Threshold* Science 279. 5350, (502 – 503)

Molofee J. 2000 OpenGL initialization source code for windows http://www.yov408.com/html/codespot.php?gg=38 accessed 31st of august 2005

Paige, K. N., and Whitham T.G.. 1987 *Overcompensation in response to mammalian herbivory: the advantage of being eaten* American Naturalist 129. 3, (407 – 416)

Parsons D.J., Armstrong A.C., Tunrpenny J.R., Matthews A.M., Copper K. and Clark J.A. 2001 *Integrated models of livestock systems for climate change studies* Global Change Biology 7, (93-112)

Randall D., Burggren W. and French K. 1997 *Eckert Animal Physiology Mechanisms and Adaptations* W.H. Freeman and Company Fourth ed.

Sitch, S., Smith, B., Prentice, I.C., Arneth, A., Bondeau, A., Cramer, W., Kaplan, J.O., Levis, S., Lucht, W., Sykes, M.T., Thonicke, K., Venevsky, S. 2003 *Evaluation of ecosystem dynamics, plant geography and terrestrial carbon cycling in the LPJ dynamic global vegetation model* Global Change Biol. 9. 2, (161 – 185)

Smith B. Prentice I.C, Sykes M. T. 2001 *Representation of vegetation dynamics in the modelling of terrestrial ecosystems: comparing two contrasting approaches within European climate space* Global Ecology & Biogeography 10. 6, (621-637)

Townsend, C.R., Harper J.L., and Begon M. 2000 *Essentials of Ecology* Blackwell Science

USEPA (United States Environmental Protection Agency) 1995 Royal Farms 175 kW Covered Lagoon System, Office of Air and Radiation, Washington D.C., August 1995 referenced at http://www.fao.org/WAIRDOCS/LEAD/X6116E/x6116e02.htm (accessed 30th of January 2006)

Yodzis P. 1989 *Introduction to Theoretical Ecology* Harper and Row

Appendix I

// Source code file name: canexch.cpp
// Written by: Ben Smith
// Version dated: 2002-12-16
* No changes

No changes

// Source code file name: driver.cpp
// Written by: Ben Smith
// Version dated: 2002-12-16

1 Change

(---) At line 607

```
// Jakob
fluxes.acflux_ruminants=0.0;
fluxes.amethan_flux=0.0;
```

// Source code file name: growth.cpp
// Written by: Ben Smith
// Version dated: 2002-12-16

2 changes

(---) At line 303

```
// Jakob
void grazing(Patch& patch,Individual& indiv,double& anpp) {

                //                  const double an_c_frac=0.45; // fraction of carbon in plant dry  matter
                //                  const double an_drym_frac=0.2 // fraction of dry animal mass in life wieght
                // const double d=0.8;  // water content of herbivore
                // const double Lc=l*d*e; // carbon fraction of herbivores
                // Hc=1.38; // Big voodoo here, but based on herbivores / m2 in europe will be calculated in later versions from input

                const double poss_int_par=0.06; // possible intake factor depending on metabolic mass
                const double plant_c_frac=0.45; // carbon content of herbivore dry matter
                const double met_mass_const=0.75; //
                const double l=250;                                     // hebivor lifewieght as defined by klieber 1967

                const double veg_quality=0.6; //nutirtional quality of vegitation
                const double urine_frac=0.04;           //fraction of digested carbon lost via urine
                const double methane_frac=0.08;                         //fraction of digested carbon lost via methane
                const double maint_cost_par=0.012;      //Cost of maintanance;
                const double herb_frac_water=0.8;
                const double herb_frac_c=0.45;
                const double growth_resp_par=0.5;

                double area[]={0, 2835878.59, 4264445.07, 1225292.14, 10010464.75, 4416719.95, 8325916.23 }; // array of bellow
                double herb[]={0, 198820113, 387143568, 359733571, 612757970, 147723464, 1303553185 };
```

```
                                        // array of bellow (1996)

                //const double continent_1=2835878.59; // sqkm size of grazzed lands in North America
                //const double continent_2=4264445.07; // sqkm size of grazzed lands in South America
                //const double continent_3=1225292.14; // sqkm size of grazzed lands in Europe
                //const double continent_4=10010464.75; // sqkm size of grazzed lands in Africa
                //const double continent_5=4416719.95; // sqkm size of grazzed lands in Austrailia
                //const double continent_6=8325916.23; // sqkm size of grazzed lands in Asia

                //const double herb_1=192920255; // size of grazzing herbivoure populations in North America
(2004 FAO)
                //const double herb_2=416611656; // size of grazzing herbivoure populations in South America
(2004 FAO)
                //const double herb_3=291510627; // size of grazzing herbivoure populations in Europe (2004
FAO)
                //const double herb_4=725028997; // size of grazzing herbivoure populations in Africa (2004
FAO)
                //const double herb_5=121320000; // size of grazzing herbivoure populations in Austrailia (2004
FAO)
                //const double herb_6=1382657964; // size of grazzing herbivoure populations in Asia (2004
FAO)

                //const double herb_1=198820113; // size of grazzing herbivoure populations in North America
(1996 FAO)
                //const double herb_2=387143568; // size of grazzing herbivoure populations in South America
(1996 FAO)
                //const double herb_3=359733571; // size of grazzing herbivoure populations in Europe (1996
FAO)
                //const double herb_4=612757970; // size of grazzing herbivoure populations in Africa (1996
FAO)
                //const double herb_5=147723464; // size of grazzing herbivoure populations in Austrailia (1996
FAO)
                //const double herb_6=1303553185; // size of grazzing herbivoure populations in Asia (2004
FAO)

                double dens;    // herbivore density in number of animals/m2
                double life_w_c;
                double met_mass;
                double met_mass_m2;
                double Ic;
                double aIc;
                double maint_cost;
                double metabolised;
                int n;                                                  //heleper variable to dump
values into
                // Thomas
                double urine_fluxC,faeces_fluxC;
                        // total production assumed to be decomposed immedeatly, not considering
faeces yet.
                double growth_resp;
                double meat_production;
                        // kg per year and m2

                // Thomas: for writing meat production to file
                FILE *Fp4;
```

```
		Fp4=fopen("meat_production.txt","a");

		if (patch.stand.grazing_continent>0){

				//life wieght to g carbon

				//		life_w_c=l*(1-herb_frac_water)*herb_frac_c;

				// metabolic mass is f(l)

				met_mass=pow(l,met_mass_const);

				// metabolic mass/m2 is f(l,herbivore_density)

				n=patch.stand.grazing_continent;		//get value for continent the gird cell is represented on
				dens=herb[n]/(area[n]*1000000);		//calculate herbivore density for the continent, recalculate from sqkm above to sqm by *1000000

				met_mass_m2=met_mass*dens;

				// Daily intake is f(poss_int_par,metabolic mass/m2)

				Ic=poss_int_par*plant_c_frac*met_mass_m2;  // Possible herbivore intake per day

				// rotine is only called once per year, so to calculate anual intake * with days of the year

				aIc=Ic*365;
							// anual intake (qaulitative assumption)

		}

		if (date.year<300+20 |! ifgrazing_global)
				aIc=0.0;

		if (patch.stand.ifgrazing==1) {

				if (aIc>anpp+indiv.cmass_leaf+indiv.cmass_root && date.year>300+50) {
					dprintf("%70s%6.2f%6.2f%5d%10.4f\n","!!Herbivore consumption higher than available biomass at ",
		patch.stand.climate.lon,patch.stand.climate.lat,date.year,
						aIc-anpp-indiv.cmass_leaf-indiv.cmass_root);
					aIc=anpp+0.9*(indiv.cmass_leaf+indiv.cmass_root);
				}
				else if (aIc>anpp) {
					// indiv.cmass_leaf-=(indiv.ltor*(bminc-aIc));
					// indiv.cmass_root-=((1.0-indiv.ltor)*(bminc-aIc));
					// Thomas
					indiv.cmass_leaf-=(indiv.ltor/(1.0+indiv.ltor)*(aIc-anpp));
					indiv.cmass_root-=(1.0/(1.0+indiv.ltor)*(aIc-anpp));
					anpp=0.0;
				}
				else
```

```
                                        anpp-=aIc;

                                maint_cost=maint_cost_par*met_mass_m2*365.0;
                                        // Thomas: mulitplied with number of days in year using
me_mass_m2
                                        // instead of met_mass
                                metabolised=aIc*veg_quality*(1.0-urine_frac-methane_frac);
                                        // per year; using aIc, not Ic
                                growth_resp=(1.0-growth_resp_par)*(metabolised-maint_cost);
                                urine_fluxC=aIc*veg_quality*urine_frac;
                                meat_production=growth_resp*2.0*1.0/herb_frac_water*1.0/herb_frac_c;

                                if (date.year>400+67)

                        fprintf(Fp4,"%6.1f%6.1f%5d%10.7f\n",patch.stand.climate.lon,patch.stand.climate.lat,
                                        date.year,meat_production);

                                if (date.year>300) {
                                        // patch.fluxes.acflux_ruminants=(aIc*(1-
veg_quality))+(aIc*veg_quality*urine_frac)+maint_cost;
                                        patch.fluxes.acflux_ruminants=metabolised+urine_fluxC; //
Thomas
                                        // faeces+urine+maintenance respiration
                                        patch.fluxes.amethan_flux=aIc*veg_quality*methane_frac;
// methane production in g C
                                }
                                else {
                                        patch.fluxes.acflux_ruminants=0.0;
                                        patch.fluxes.amethan_flux=0.0;
                                }

                        }
fclose(Fp4);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/

(---) At line 1080

                                        // Jakob (comment below for run without grazing)
                                        // Thomas: called before reproduction and with indiv.anpp
instead of bminc
                                        if (ifgrazing_global)
                                                grazing(patch,indiv,indiv.anpp);
```

// Source code file name: canexch.cpp
// Written by: Ben Smith
// Version dated: 2002-12-16

No changes

* No changes

// Source code file name: guess.cpp
// Written by: Ben Smith
// Version dated: 2002-12-16

No changes

* No changes

// Source code file name: driver.cpp
// Written by: Ben Smith
// Version dated: 2002-12-16

7 Changes

(---) At line 799

```
// Jakob
FILE *in_grazing;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///
```

(---) At line 993

```
// Jakob
		xtring file_grazing;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///
```

(---) At line 1072

```
// Jakob
		file_grazing=param["file_grazing"].str;
		in_grazing=fopen(file_grazing,"rt");
		if (!in_grazing) fail("initio: could not open %s for input",(char*)file_grazing);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///
```

(---) At line 1152

```
// Jakob
		double lon_grazing,lat_grazing;

		if (firstgrid) {
				gridlist.firstobj();
				firstgrid=false;
		}
		else gridlist.nextobj();

		if (gridlist.isobj) {

				// Load environmental data for this grid cell from CRU file
				gridfound=searchcru(in_cru,gridlist.getobj().lon,

		gridlist.getobj().lat,soilcode,hist_mtemp,hist_mprec,hist_msun);

				// Jakob
				stand.climate.lon=gridlist.getobj().lon;

		readfor(in_grazing,"f,f,i,f",&lon_grazing,&lat_grazing,&stand.ifgrazing,&stand.grazing_conti
nent);
```

Appendix II

Data pre processing

```
// Appendix 2(1)
// Written by:          Jakob Lagerstedt
// Version dated:       2005-07
#include <stdio.h> // standard input/output libary in C++
#include <time.h> // used in gutil.h
#include <string.h> // used in gutil.h
#include <gutil.h> // Ben's input/output utility

// Specified by user:
// does not work for alst grid cell

int main() {

                                        double lon;
                                        double lat;
                                        int value;
                                        int values;

                                        lon=lat=0.0;


   FILE * in_data = fopen("llgrazing_cru.txt","rt"); // rt = read text

   FILE * out_data = fopen("dcontllgrazing_cru.txt","wt"); // wt = write text

                                        if(!in_data) {
                                                printf("Could not open input\n");
                                                return 0;
                                        }

                                        if(!out_data) {
                                                printf("Could not open output\n");
                                                return 0;
                                        }

                                for (int n=0;n<59191;n++)
                                  // while (!feof(in_data))
                                                {
                                                readfor(in_data,"f,f,i,i",&lon,&lat,&value,&values);



                                                                                        //
fprintf(out_data,"%8.2f%8.2f%5d%5d\n",lon,lat,value,values);

                                                                                        if (lat<90&& lat>12&& lon>-
180&& lon<-40)

                fprintf(out_data,"%8.2f%8.2f%5d%5d\n",lon,lat,value,1);
                                                                                        else if (lat<=12&& lat>-90&&
lon>-180&& lon<-30)
```

```
            fprintf(out_data,"%8.2f%8.2f%5d%5d\n",lon,lat,value,2);
                                                                    else if (lat<90&& lat>35&&
lon>-30&& lon<35)
            fprintf(out_data,"%8.2f%8.2f%5d%5d\n",lon,lat,value,3);
                                                                    else if (lat<=35&& lat>-90&&
lon>-30&& lon<54)
            fprintf(out_data,"%8.2f%8.2f%5d%5d\n",lon,lat,value,4);
                                                                    else if (lat<0&& lat>-90&&
lon>80&& lon<180)
            fprintf(out_data,"%8.2f%8.2f%5d%5d\n",lon,lat,value,5);
                                                                    else

            fprintf(out_data,"%8.2f%8.2f%5d%5d\n",lon,lat,value,6);


                                    }

                                fclose(in_data);

                                fclose(out_data);
                                    return 0;
}
```

// Appendix 2(2)
// Written by: Jakob Lagerstedt
// Version dated: 2005-08

Visualization application

```
// Window handling by Jeff Molofee 2000

#include <windows.h>                    //Headers
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glaux.h>
#include <stdio.h> // standard input/output libary in C++
#include <time.h> // used in gutil.h
#include <string.h> // used in gutil.h
#include <gutil.h> // Ben's input/output utility

HDC                                     hDC=NULL;
HGLRC                   hRC=NULL;
HWND                    hWnd=NULL;
HINSTANCE   hInstance;

bool            keys[256];
bool            active=TRUE;
bool            fullscreen=TRUE;

LRESULT     CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

GLvoid ReSizeGLScene(GLsizei width, GLsizei height)
            if (height==0)
```

```
		{
			height=1;

		}

		glViewport(0,0,width,height);


		glMatrixMode(GL_PROJECTION);

		glLoadIdentity();


		// Calculate The Aspect Ratio Of The Window
		gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,400.0f);

		glMatrixMode(GL_MODELVIEW);

		glLoadIdentity();

}

int InitGL(GLvoid)

{
		glShadeModel(GL_SMOOTH);
							// Enable Smooth Shading
		glClearColor(0.0f, 0.0f, 0.0f, 0.5f);
		// Black Background
		glClearDepth(1.0f);
									// Depth Buffer Setup
		glEnable(GL_DEPTH_TEST);
							// Enables Depth Testing
		glDepthFunc(GL_LEQUAL);
								// The Type Of Depth Testing To Do
		glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);		// Perspective
Calculations
		return TRUE;
									// Initialization Went OK
}

int DrawGLScene(GLvoid)
								// Here's Where We Do All The Drawing
{

		double lon;

		double lat;
		int value;
		int values;
		int cont;
		lon=lat=25.0;

		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
		glLoadIdentity();
```

```
		FILE * in_data = fopen("dcontllgrazing_cru.txt","rt"); // rt = read text
		if(!in_data) {
				printf("Could not open input\n");
				return 0;
		}

		while (!feof(in_data))
				{
				readfor(in_data,"f,f,i,i",&lon,&lat,&value,&values);
				glLoadIdentity();
                                                                // Reset The Current
Modelview Matrix
//				lat=lat/5;
                                                                        // Variable
Voodoo
//				lon=lon/5;
				glTranslatef(lon-20,lat,-220.0f);
                                        // Move to lat long position at z 100

				if (value==1 && values==1)
                                                {
                                                                glBegin(GL_QUADS);

                                        // Draw A Quad
                                                                glColor3f(1.0f,1.0f,1.0f);

		// Colour
                                                                glVertex3f(-0.1f, 0.1f, 0.0f);
                                                                        // Top Left
                                                                glVertex3f( 0.1f, 0.1f, 0.0f);
                                                                        // Top Right
                                                                glVertex3f( 0.1f,-0.1f, 0.0f);
                                                                        // Bottom
Right
                                                                glVertex3f(-0.1f,-0.1f, 0.0f);
                                                                        // Bottom Left
                                                                glEnd();

                                                // Done Drawing The Quad
                                                }

				if (value==1 && values==2)
                                                {
                                                                glBegin(GL_QUADS);

                                        // Draw A Quad
                                                                glColor3f(1.0f,0.0f,1.0f);

		// Colour
                                                                glVertex3f(-0.1f, 0.1f, 0.0f);
                                                                        // Top Left
                                                                glVertex3f( 0.1f, 0.1f, 0.0f);
                                                                        // Top Right
                                                                glVertex3f( 0.1f,-0.1f, 0.0f);
                                                                        // Bottom
Right
```

```
                                        glVertex3f(-0.1f,-0.1f, 0.0f);
                                                // Bottom Left
                                        glEnd();

                                // Done Drawing The Quad
                                }

                        if (value==1 && values==3)
                                {
                                        glBegin(GL_QUADS);

                                // Draw A Quad
                                        glColor3f(1.0f,1.0f,0.0f);

        // Colour
                                        glVertex3f(-0.1f, 0.1f, 0.0f);
                                                // Top Left
                                        glVertex3f( 0.1f, 0.1f, 0.0f);
                                                // Top Right
                                        glVertex3f( 0.1f,-0.1f, 0.0f);
                                                // Bottom
Right
                                        glVertex3f(-0.1f,-0.1f, 0.0f);
                                                // Bottom Left
                                        glEnd();

                                // Done Drawing The Quad
                                }

                        if (value==1 && values==4)
                                {
                                        glBegin(GL_QUADS);

                                // Draw A Quad
                                        glColor3f(1.0f,0.0f,0.0f);

        // Colour
                                        glVertex3f(-0.1f, 0.1f, 0.0f);
                                                // Top Left
                                        glVertex3f( 0.1f, 0.1f, 0.0f);
                                                // Top Right
                                        glVertex3f( 0.1f,-0.1f, 0.0f);
                                                // Bottom
Right
                                        glVertex3f(-0.1f,-0.1f, 0.0f);
                                                // Bottom Left
                                        glEnd();

                                // Done Drawing The Quad
                                }

                        if (value==1 && values==6)
                                {
                                        glBegin(GL_QUADS);

                                // Draw A Quad
```

```
                                                glColor3f(0.0f,0.0f,1.0f);

        // Colour
                                                glVertex3f(-0.1f, 0.1f, 0.0f);
                                                        // Top Left
                                                glVertex3f( 0.1f, 0.1f, 0.0f);
                                                        // Top Right
                                                glVertex3f( 0.1f,-0.1f, 0.0f);
                                                        // Bottom
Right
                                                glVertex3f(-0.1f,-0.1f, 0.0f);
                                                        // Bottom Left
                                                glEnd();

                                        // Done Drawing The Quad
                                        }

                        if (value==1 && values==5)
                                        {
                                                glBegin(GL_QUADS);

                                // Draw A Quad
                                                glColor3f(0.0f,1.0f,0.0f);

        // Colour
                                                glVertex3f(-0.1f, 0.1f, 0.0f);
                                                        // Top Left
                                                glVertex3f( 0.1f, 0.1f, 0.0f);
                                                        // Top Right
                                                glVertex3f( 0.1f,-0.1f, 0.0f);
                                                        // Bottom
Right
                                                glVertex3f(-0.1f,-0.1f, 0.0f);
                                                        // Bottom Left
                                                glEnd();

                                        // Done Drawing The Quad
                                        }

        }
        fclose(in_data);
        return TRUE;

}

GLvoid KillGLWindow(GLvoid)
                                //Kill WWndow
{
        if (fullscreen)

        {
                ChangeDisplaySettings(NULL,0);
                ShowCursor(TRUE);
                                                // Show Mouse Pointer
        }
```

```
		if (hRC)
		{
			if (!wglMakeCurrent(NULL,NULL))

			{
				MessageBox(NULL,"Release Of DC And RC Failed.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
			}

			if (!wglDeleteContext(hRC))

			{
				MessageBox(NULL,"Release Rendering Context Failed.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
			}
			hRC=NULL;

		}

		if (hDC && !ReleaseDC(hWnd,hDC))
		{
			MessageBox(NULL,"Release Device Context Failed.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
			hDC=NULL;
		}

		if (hWnd && !DestroyWindow(hWnd))
			// Are We Able To Destroy The Window?
		{
			MessageBox(NULL,"Could Not Release hWnd.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
			hWnd=NULL;

		}

		if (!UnregisterClass("OpenGL",hInstance))
		{
			MessageBox(NULL,"Could Not Unregister Class.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
			hInstance=NULL;

		}
}

BOOL CreateGLWindow(char* title, int width, int height, int bits, bool fullscreenflag)
{
		GLuint		PixelFormat;		// Holds The Results After Searching For A Match
		WNDCLASS	wc;
			// Windows Class Structure
		DWORD		dwExStyle;
		// Window Extended Style
		DWORD		dwStyle;
		// Window Style
```

```
	RECT			WindowRect;
	// Grabs Rectangle Upper Left / Lower Right Values
	WindowRect.left=(long)0;				// Set Left Value To 0
	WindowRect.right=(long)width;		// Set Right Value To Requested Width
	WindowRect.top=(long)0;					// Set Top Value To 0
	WindowRect.bottom=(long)height;			// Set Bottom Value To Requested Height

	fullscreen=fullscreenflag;				// Set The Global Fullscreen Flag

	hInstance			= GetModuleHandle(NULL);

	wc.style			= CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
	wc.lpfnWndProc			= (WNDPROC) WndProc;

	wc.cbClsExtra		= 0;

	wc.cbWndExtra		= 0;

	wc.hInstance		= hInstance;

	wc.hIcon			= LoadIcon(NULL, IDI_WINLOGO);

	wc.hCursor			= LoadCursor(NULL, IDC_ARROW);

	wc.hbrBackground	= NULL;

	wc.lpszMenuName			= NULL;


	wc.lpszClassName	= "OpenGL";


	if (!RegisterClass(&wc))
						// Attempt To Register The Window Class
	{
		MessageBox(NULL,"Failed To Register The Window Class.","ERROR",MB_OK|MB_ICONEXCLAMATION);
		return FALSE;

	// Return FALSE
	}

	if (fullscreen)


	{
		DEVMODE dmScreenSettings;

		memset(&dmScreenSettings,0,sizeof(dmScreenSettings));
		dmScreenSettings.dmSize=sizeof(dmScreenSettings);
```

```
                        dmScreenSettings.dmPelsWidth= width;

                        dmScreenSettings.dmPelsHeight                   = height;

                        dmScreenSettings.dmBitsPerPel= bits;


        dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;

                        // Try To Set Selected Mode And Get Results.  NOTE: CDS_FULLSCREEN
Gets Rid Of Start Bar.
                        if
(ChangeDisplaySettings(&dmScreenSettings,CDS_FULLSCREEN)!=DISP_CHANGE_SUCCESSFUL)
                        {
                                if (MessageBox(NULL,"The Requested Fullscreen Mode Is Not
Supported By\nYour Video Card. Use Windowed Mode Instead?","NeHe
GL",MB_YESNO|MB_ICONEXCLAMATION)==IDYES)
                                {
                                        fullscreen=FALSE;
        // Windowed Mode Selected.  Fullscreen = FALSE
                                }
                                else
                                {
                                        // Pop Up A Message Box Letting User Know
The Program Is Closing.
                                        MessageBox(NULL,"Program Will Now
Close.","ERROR",MB_OK|MB_ICONSTOP);
                                        return FALSE;

        // Return FALSE
                                }
                        }
        }

        if (fullscreen)


                        dwExStyle=WS_EX_APPWINDOW;

                        dwStyle=WS_POPUP;


                        ShowCursor(FALSE);


        }
        else
        {
                        dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;

                        dwStyle=WS_OVERLAPPEDWINDOW;

        }

        AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle);
```

```
// Create The Window
if (!(hWnd=CreateWindowEx(  dwExStyle,

                        "OpenGL",
                                        // Class Name

                        title,

                        dwStyle |

                        WS_CLIPSIBLINGS |

                        WS_CLIPCHILDREN,

                        0, 0,

                        WindowRect.right-WindowRect.left,

                        WindowRect.bottom-WindowRect.top,

                        NULL,

                        NULL,

                        hInstance,

                        NULL)))

{
                        KillGLWindow();
                                                                // Reset The Display
                        MessageBox(NULL,"Window Creation
Error.","ERROR",MB_OK|MB_ICONEXCLAMATION);
                        return FALSE;
                                                // Return FALSE
}

static          PIXELFORMATDESCRIPTOR pfd=

{
                        sizeof(PIXELFORMATDESCRIPTOR),

                        1,

                        PFD_DRAW_TO_WINDOW |
```

```
                PFD_SUPPORT_OPENGL |
                PFD_DOUBLEBUFFER,
                PFD_TYPE_RGBA,
                bits,
                0, 0, 0, 0, 0, 0,
                0,
                0,
                0,
                0, 0, 0, 0,
                16,
                0,
                0,
                PFD_MAIN_PLANE,
                0,
                0, 0, 0
        };
        if (!(hDC=GetDC(hWnd)))
        {
                KillGLWindow();
                                                        // Reset The Display
                MessageBox(NULL,"Can't Create A GL Device
Context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
                return FALSE;
                                                // Return FALSE
        }
        if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd)))
        {
                KillGLWindow();
                                                        // Reset The Display
                MessageBox(NULL,"Can't Find A Suitable
```

```
PixelFormat.","ERROR",MB_OK|MB_ICONEXCLAMATION);
                                return FALSE;
                                                                // Return FALSE
                }

                if(!SetPixelFormat(hDC,PixelFormat,&pfd))
                {
                                KillGLWindow();
                                                                // Reset The Display
                                MessageBox(NULL,"Can't Set The
PixelFormat.","ERROR",MB_OK|MB_ICONEXCLAMATION);
                                return FALSE;
                                                                // Return FALSE
                }

                if (!(hRC=wglCreateContext(hDC)))
                // Are We Able To Get A Rendering Context?
                {
                                KillGLWindow();
                                                                // Reset The Display
                                MessageBox(NULL,"Can't Create A GL Rendering
Context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
                                return FALSE;
                                                                // Return FALSE
                }

                if(!wglMakeCurrent(hDC,hRC))

                {
                                KillGLWindow();
                                                                // Reset The Display
                                MessageBox(NULL,"Can't Activate The GL Rendering
Context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
                                return FALSE;
                                                                // Return FALSE
                }

                ShowWindow(hWnd,SW_SHOW);

                SetForegroundWindow(hWnd);
                SetFocus(hWnd);

                ReSizeGLScene(width, height);

                if (!InitGL())
                                                                // Initialize Our Newly Created GL Window
                {
                                KillGLWindow();
                                                                // Reset The Display
                                MessageBox(NULL,"Initialization
Failed.","ERROR",MB_OK|MB_ICONEXCLAMATION);
                                return FALSE;
                                                                // Return FALSE
                }

                return TRUE;
```

```
}
LRESULT CALLBACK WndProc(		HWND		hWnd,

		UINT		uMsg,

		WPARAM		wParam,

		LPARAM		lParam)
{
		switch (uMsg)

		{
			case WM_ACTIVATE:

			{
				if (!HIWORD(wParam))

				{
					active=TRUE;

				}
				else
				{
					active=FALSE;

				}

				return 0;

			}

			case WM_SYSCOMMAND:
			{
				switch (wParam)
				{
					case SC_SCREENSAVE:
					case SC_MONITORPOWER:
						return 0;
				}
				break;
			}

			case WM_CLOSE:

			{
				PostQuitMessage(0);
					// Send A Quit Message
				return 0;

			}

			case WM_KEYDOWN:
					// Is A Key Being Held Down?
			{
```

```
                                        keys[wParam] = TRUE;
                                        // If So, Mark It As TRUE
                                        return 0;
                                                                // Jump Back
                                }

                                case WM_KEYUP:
                                                                // Has A Key Been Released?
                                {
                                        keys[wParam] = FALSE;
                                        // If So, Mark It As FALSE
                                        return 0;
                                                                // Jump Back
                                }

                                case WM_SIZE:

                                {
                                        ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));
                                        return 0;
                                                                // Jump Back
                                }
                        }

                        // Pass All Unhandled Messages To DefWindowProc
                        return DefWindowProc(hWnd,uMsg,wParam,lParam);
}

int WINAPI WinMain(             HINSTANCE   hInstance,
                                                        HINSTANCE   hPrevInstance,

                                                        LPSTR
                        lpCmdLine,
                                                        int
                                nCmdShow)
{
                        MSG                     msg;

                        BOOL            done=FALSE;


                        // Ask The User Which Screen Mode They Prefer
//                      if (MessageBox(NULL,"Fullscreen Mode?", "Start
FullScreen?",MB_YESNO|MB_ICONQUESTION)==IDNO)
//                      {
                                fullscreen=FALSE;

//                      }

                        // Create Our OpenGL Window
                        if (!CreateGLWindow("Dataview LJP OpenGL",1400,800,16,fullscreen))
                        {
                                return 0;

                        }
```

```
	while(!done)

	{
		if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))
		{
			if (msg.message==WM_QUIT)

			{
				done=TRUE;

			}
			else

			{
				TranslateMessage(&msg);

				DispatchMessage(&msg);

			}
		}
		else

		{
			// Draw The Scene.  Watch For ESC Key And Quit Messages
From DrawGLScene()
			if ((active && !DrawGLScene()) || keys[VK_ESCAPE])
	// Active?  Was There A Quit Received?
			{
				done=TRUE;

			}
			else

			{
				SwapBuffers(hDC);
			}

			if (keys[VK_F1])
			{
				keys[VK_F1]=FALSE;

				KillGLWindow();

				fullscreen=!fullscreen;

				// Recreate Our OpenGL Window
				if (!CreateGLWindow("Dataview LJP
OpenGL",1200,860,16,fullscreen))
				{
					return 0;

				}
			}
		}
	}
```

```
		// Shutdown
		KillGLWindow();
                                                                        // Kill The Window
		return (msg.wParam);
                                                // Exit The Program
}
```

// Appendix 2(3) Data post processing
// Written by: Jakob Lagerstedt
// Version dated: 2005-08

```
#include <stdio.h> // standard input/output libary in C++
#include <time.h> // used in gutil.h
#include <string.h> // used in gutil.h
#include <gutil.h> // Ben's input/output utility
#include <stdlib.h>
#include <iostream.h>
#include <math.h>

// calculates grazed area per continent

double pixelsize(double longpos,double latpos,double longsize,double latsize,int postype) {

//c    (Ben Smith, 15/5/97)
//
//c    Returns area in square km of a pixel of a given size at a given point
//c    on the world.  The formula applied is the surface area of a segment of
//c    a hemisphere of radius r from the equator to a parallel (circular)
//c    plane h vertical units towards the pole: S=2*pi*r*h
//
//c    longpos   longitude position (see postype)
//c    latpos    latitude position (see postype)
//c    longsize  longitude range in degrees
//c    latsize   latitude range in degrees
//c    postype   declares which part of the pixel longpos and latpos
//c              refer to:
//c              0 = centre
//c              1 = NW corner
//c              2 = NE corner
//c              3 = SW corner
//c              4 = SE corner

                double pi,r,h1,h2,lattop,latbot,s;

                pi=3.1415926536;
                r=6367.425;   //mean radius of the earth

                lattop=latpos;
                if (postype==0) lattop=latpos+latsize*0.5;
                if (postype==3 || postype==4) lattop=latpos+latsize;
                if (lattop<0.0) lattop=-lattop+latsize;
                latbot=lattop-latsize;
                h1=r*sin(lattop*pi/180.0);
                h2=r*sin(latbot*pi/180.0);
                s=2.0*pi*r*(h1-h2);  //for this latitude band
```

```
		return s*longsize/360.0;  //for this pixel
}



int main() {

		double lon;
		double lat;
		int n;
		int year;
		int year_set;

		double trbe;
		double trbr;
		double tene;
		double tebe;
		double tebs;
		double bne;
		double bs;
		double c3g;
		double c4g;
		double npp;
		double npp_area_adjusted;
		static double world_map[720][360];
		double out;
		int a;
		int b;
		int c;
		int d;
		int e;

		double pixelarea;
		int i;


		npp=out=npp_area_adjusted=lon=lat=0.0;

		a=0;
		b=0;
		for (a=0; a<720; a++){
				for (b=0; b<360; b++){
						world_map[a][b]=-9999.0;
				}
		}


		cout << world_map[719][359];
		cout << world_map[0][0];
		cout << "\n";

//		year_set=370;
//		n=year_set-370;

//		cout << "skriv in vilket Âr du vill bˆrja ber‰kna 370-397 ";
```

```
//                  cin >> year_set;

                    cout << "automode 370-397 ";

                    FILE * out_data = fopen("NEE.G.OFF.xxx.ASCIITORASTER.txt","wt"); // wt = write text
                    fclose(out_data);

                    for (year_set=370; year_set<397; year_set++){


                                        if (year_set==396){
                                                                FILE * out_data =
fopen("NPP.G.ON.396.ASCIITORASTER.txt","wt"); // wt = write text
                                                                                    if(!out_data) {
                                                                                                        printf("Could
not open output\n");
                                                                                                        return 0;
                                                                                                        }
                                        }

                                        if (year_set==395){
                                                                FILE * out_data =
fopen("NPP.G.ON.395.ASCIITORASTER.txt","wt"); // wt = write text
                                                                                    if(!out_data) {
                                                                                                        printf("Could
not open output\n");
                                                                                                        return 0;
                                                                                                        }
                                        }

                                        if (year_set==394){
                                                                FILE * out_data =
fopen("NPP.G.ON.394.ASCIITORASTER.txt","wt"); // wt = write text
                                                                                    if(!out_data) {
                                                                                                        printf("Could
not open output\n");
                                                                                                        return 0;
                                                                                                        }
                                        }

                                        if (year_set==393){
                                                                FILE * out_data =
fopen("NPP.G.ON.393.ASCIITORASTER.txt","wt"); // wt = write text
                                                                                    if(!out_data) {
                                                                                                        printf("Could
not open output\n");
                                                                                                        return 0;
                                                                                                        }
                                        }

                                        if (year_set==392){
                                                                FILE * out_data =
fopen("NPP.G.ON.392.ASCIITORASTER.txt","wt"); // wt = write text
                                                                                    if(!out_data) {
                                                                                                        printf("Could
not open output\n");
```

```
                                                                    return 0;
                                                                    }
                                        }

                                        if (year_set==391){
                                                                FILE * out_data =
fopen("NPP.G.ON.391.ASCIITORASTER.txt","wt"); // wt = write text
                                                                if(!out_data) {
                                                                    printf("Could
not open output\n");
                                                                    return 0;
                                                                    }
                                        }

                                        if (year_set==390){
                                                                FILE * out_data =
fopen("NPP.G.ON.390.ASCIITORASTER.txt","wt"); // wt = write text
                                                                if(!out_data) {
                                                                    printf("Could
not open output\n");
                                                                    return 0;
                                                                    }
                                        }

                                        if (year_set==389){
                                                                FILE * out_data =
fopen("NPP.G.ON.389.ASCIITORASTER.txt","wt"); // wt = write text
                                                                if(!out_data) {
                                                                    printf("Could
not open output\n");
                                                                    return 0;
                                                                    }
                                        }

                                        if (year_set==388){
                                                                FILE * out_data =
fopen("NPP.G.ON.388.ASCIITORASTER.txt","wt"); // wt = write text
                                                                if(!out_data) {
                                                                    printf("Could
not open output\n");
                                                                    return 0;
                                                                    }
                                        }

                                        if (year_set==387){
                                                                FILE * out_data =
fopen("NPP.G.ON.387.ASCIITORASTER.txt","wt"); // wt = write text
                                                                if(!out_data) {
                                                                    printf("Could
not open output\n");
                                                                    return 0;
                                                                    }
                                        }

                                        if (year_set==386){
                                                                FILE * out_data =
```

```
fopen("NPP.G.ON.386.ASCIITORASTER.txt","wt"); // wt = write text
                                                                        if(!out_data) {
                                                                                printf("Could
not open output\n");
                                                                                return 0;
                                                                                }
                                        }

                                        if (year_set==385){
                                                                FILE * out_data =
fopen("NPP.G.ON.385.ASCIITORASTER.txt","wt"); // wt = write text
                                                                        if(!out_data) {
                                                                                printf("Could
not open output\n");
                                                                                return 0;
                                                                                }
                                        }

                                        if (year_set==384){
                                                                FILE * out_data =
fopen("NPP.G.ON.384.ASCIITORASTER.txt","wt"); // wt = write text
                                                                        if(!out_data) {
                                                                                printf("Could
not open output\n");
                                                                                return 0;
                                                                                }
                                        }

                                        if (year_set==383){
                                                                FILE * out_data =
fopen("NPP.G.ON.383.ASCIITORASTER.txt","wt"); // wt = write text
                                                                        if(!out_data) {
                                                                                printf("Could
not open output\n");
                                                                                return 0;
                                                                                }
                                        }

                                        if (year_set==382){
                                                                FILE * out_data =
fopen("NPP.G.ON.382.ASCIITORASTER.txt","wt"); // wt = write text
                                                                        if(!out_data) {
                                                                                printf("Could
not open output\n");
                                                                                return 0;
                                                                                }
                                        }
                                        if (year_set==381){
                                                                FILE * out_data =
fopen("NPP.G.ON.381.ASCIITORASTER.txt","wt"); // wt = write text
                                                                        if(!out_data) {
                                                                                printf("Could
not open output\n");
                                                                                return 0;
                                                                                }
                                        }
```

```
                    if (year_set==380){
                                        FILE * out_data =
fopen("NPP.G.ON.380.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }

                    if (year_set==379){
                                        FILE * out_data =
fopen("NPP.G.ON.379.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }
                    if (year_set==378){
                                        FILE * out_data =
fopen("NPP.G.ON.378.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }

                    if (year_set==377){
                                        FILE * out_data =
fopen("NPP.G.ON.377.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }
                    if (year_set==376){
                                        FILE * out_data =
fopen("NPP.G.ON.376.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }

                    if (year_set==375){
                                        FILE * out_data =
fopen("NPP.G.ON.375.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
```

```
                    }

                    if (year_set==374){
                                        FILE * out_data =
fopen("NPP.G.ON.374.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }

                    if (year_set==373){
                                        FILE * out_data =
fopen("NPP.G.ON.373.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }

                    if (year_set==372){
                                        FILE * out_data =
fopen("NPP.G.ON.372.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }

                    if (year_set==371){
                                        FILE * out_data =
fopen("NPP.G.ON.371.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }

                    if (year_set==370){
                                        FILE * out_data =
fopen("NPP.G.ON.370.ASCIITORASTER.txt","wt"); // wt = write text
                                                if(!out_data) {
                                                        printf("Could
not open output\n");
                                                        return 0;
                                                        }
                    }


          //for (int n=0;n<59191;n++)
          // fprintf(out_data,"%8.2f%8.2f%5d%5d\n",lon,lat,value,values);
```

```
		FILE * in_data = fopen("anpp.txt","rt"); // rt = read text

		if(!in_data) {
		printf("Could not open input\n");
		return 0;
		}

		while (!feof(in_data))
				{

		readfor(in_data,"f,f,i,f,f,f,f,f,f,f,f,f,f",&lon,&lat,&year,&trbe,&trbr,&tene,&tebe,&tebs,&bne,&b
s,&c3g,&c4g,&npp);

								if (year==year_set)
								{
								pixelarea=pixelsize(lon,lat,0.5,0.5,0);


		npp_area_adjusted=npp*pixelarea*1000000*1000;
				//g C NEE -- pixelsize (relative size)// * 1000000 (ideal pixel size 0.5 * 0.5
degree cell) *1000 (for conversion form kg into g)
								c=lon*2+360;
								d=lat*2+180;
								world_map[c][d]=npp_area_adjusted;
								}
				}

		for (int y=359; y>0; y--){
				for (int x=0; x<720; x++){
						out=world_map[x][y];

						fprintf(out_data,"     ");
						if (out==-9999){
								fprintf(out_data,"-9999");
						}
						else {
								fprintf(out_data,"%8.5f",out);
						}
						fprintf(out_data,"     ");
						if (x==359){
								fprintf(out_data,"\n");
						}
				}
				fprintf(out_data,"\n");
		}

		fclose(in_data);
		fclose(out_data);
		cout << "hepp \n";
		cout << year_set;

		}
		return 0;
}
```

```
                                if (lon_grazing!=gridlist.getobj().lon || lat_grazing!=gridlist.getobj().lat)
                                        dprintf("Error !!!!!!, lon lat don't match");

                                while (!gridfound) {

                                        dprintf("\nError: could not find stand at (%g,%g) in CRU data
file\n",
                                                gridlist.getobj().lon,gridlist.getobj().lat);

                                        gridlist.nextobj();
                                        if (gridlist.isobj) {

                gridfound=searchcru(in_cru,gridlist.getobj().lon,

                gridlist.getobj().lat,soilcode,hist_mtemp,hist_mprec,hist_msun);
                                        }
                                        else return false;
                                }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///

(---) At line 1338

                // Jakob
                double flux_carbon_ruminants,flux_methan_ruminants; // average over all patches, equals
stand if LPJ-DGVM

                if (date.year==0 && gridlist.getobj().id==0) {

                        // Very first time only

                        // Print column labels

                        fprintf(out_cmass,"%6s%6s%6s","Lon","Lat","Year");
                        fprintf(out_anpp,"%6s%6s%6s","Lon","Lat","Year");
                        fprintf(out_lai,"%6s%6s%6s","Lon","Lat","Year");

                fprintf(out_flux,"%6s%6s%6s%8s%8s%8s%8s%8s","Lon","Lat","Year","Veg","Soil",
                                        "Fire","Est","NEE");

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///

(---) At line 1455

                // Jakob
                        flux_carbon_ruminants=flux_methan_ruminants=0.0;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///

(---) At line 1488

// Jakob
```

```
plot("fluxes","flux_C_ruminants",date.year,flux_carbon_ruminants);

plot("fluxes","flux_M_ruminants",date.year,flux_methan_ruminants);
```

// Source code file name: main.cpp
// Written by: Ben Smith
// Version dated: 2001-09-05
* No changes

No changes

// Source code file name: soilwater.cpp
// Written by: Ben Smith
// Version dated: 2003-01-20
* No changes

No changes

// Source code file name: vegdynam.cpp
// Written by: Ben Smith
// Version dated: 2002-11-22

1 Change

(---) At line 128

```
// Jakob (comment bellow for a run without grazing both if and return)
        // Thomas: date.year>300 included
        if (patch.stand.ifgrazing==1 && pft.lifeform==TREE && date.year>300 && ifgrazing_global)
                // grazed grid cell, not establishment of woody PFTs
                return false;
        else
                return true;
}
```

Lunds Universitets Naturgeografiska institution. Seminarieuppsatser. Uppsatserna finns tillgängliga på Naturgeografiska institutionens bibliotek, Sölvegatan 12, 223 62 LUND. Serie startade 1985.

The reports are available at the Geo-Library, Department of Physical Geography, University of Lund, Sölvegatan 12, S-223 62 Lund, Sweden.
Report series started 1985.

79. Ullman, M., (2001): El Niño Southern Oscillation och dess atmosfäriska fjärrpåverkan.
80. Andersson, A., (2001): The wind climate of northwestern Europe in SWECLIM regional climate scenarios.
81. Lalloo, D., (2001): Geografiska informationssystem för studier av polyaromatiska kolväten (PAH) – Undersökning av djupvariation i BO01-området, Västra hamnen, Malmö, samt utveckling av en matematisk formel för beräkning av PAH-koncentrationer från ett kontinuerligt utsläpp.
82. Almqvist, J., Fergéus, J., (2001): GIS-implementation in Sri Lanka.
    Part 1: GIS-applications in Hambantota district Sri Lanka : a case study.
    Part 2: GIS in socio-economic planning : a case study.
83. Berntsson, A., (2001): Modellering av reflektans från ett sockerbetsbestånd med hjälp av en strålningsmodell.
84. Umegård, J., (2001): Arctic aerosol and long-range transport.
85. Rosenberg, R., (2002): Tetratermmodellering och regressionsanalyser mellan topografi, tetraterm och tillväxt hos sitkagran och lärk – en studie i norra Island.
86. Håkansson, J., Kjörling, A., (2002): Uppskattning av mängden kol i trädform – en metodstudie.
87. Arvidsson, H., (2002): Coastal parallel sediment transport on the SE Australian inner shelf – A study of barrier morphodynamics.
88. Bemark, M., (2002): Köphultssjöns tillstånd och omgivningens påverkan.
89. Dahlberg, I., (2002): Rödlistade kärlväxter i Göteborgs innerstad – temporal och rumslig analys av rödlistade kärlväxter i Göteborgs artdataarkiv, ADA.
90. Poussart, J-N., (2002): Verification of Soil Carbon Sequestration - Uncertainties of Assessment Methods.
91. Jakubaschk, C., (2002): Acacia senegal, Soil Organic Carbon and Nitrogen Contents: A Study in North Kordofan, Sudan.
92. Lindqvist, S., (2002): Skattning av kväve i gran med hjälp av fjärranalys.
93. Göthe, A., (2002): Översvämningskartering av Vombs ängar.
94. Lööv, A., (2002): Igenväxning av Köphultasjö – bakomliggande orsaker och processer.
95. Axelsson, H., (2003): Sårbarhetskartering av bekämpningsmedels läckage till grundvattnet – Tillämpat på vattenskyddsområdet Ignaberga-Hässleholm.
96. Hedberg, M., Jönsson, L., (2003): Geografiska Informationssystem på Internet – En webbaserad GIS-applikation med kalknings- och försurningsinformation för Kronobergs län.
97. Svensson, J., (2003): Wind Throw Damages on Forests – Frequency and Associated Pressure Patterns 1961-1990 and in a Future Climate Scenario.

98. Stroh, E., (2003): Analys av fiskrättsförhållandena i Stockholms skärgård i relation till känsliga områden samt fysisk störning.
99. Bäckstrand, K., (2004): The dynamics of non-methane hydrocarbons and other trace gas fluxes on a subarctic mire in northern Sweden.
100. Hahn, K., (2004): Termohalin cirkulation i Nordatlanten.
101. Lina Möllerström (2004): Modelling soil temperature & soil water availability in semi-arid Sudan: validation and testing.
102. Setterby, Y., (2004): Igenväxande hagmarkers förekomst och tillstånd i Västra Götaland.
103. Edlundh, L., (2004): Utveckling av en metodik för att med hjälp av lagerföljdsdata och geografiska informationssystem (GIS) modellera och rekonstruera våtmarker i Skåne.
104. Schubert, P., (2004): Cultivation potential in Hambantota district, Sri Lanka
105. Brage, T., (2004): Kvalitetskontroll av servicedatabasen Sisyla
106. Sjöström., M., (2004): Investigating Vegetation Changes in the African Sahel 1982-2002: A Comparative Analysis Using Landsat, MODIS and AVHRR Remote Sensing Data
107. Danilovic, A., Stenqvist, M., (2004): Naturlig föryngring av skog
108. Materia, S., (2004): Forests acting as a carbon source: analysis of two possible causes for Norunda forest site
109. Hinderson, T., (2004): Analysing environmental change in semi-arid areas in Kordofan, Sudan
110. Andersson, J., (2004): Skånska småvatten nu och då - jämförelse mellan 1940, 1980 och 2000-talet
111. Tränk, L., (2005): Kadmium i skånska vattendrag – en metodstudie i föroreningsmodellering.
112. Nilsson, E., Svensson, A.-K., (2005): Agro-Ecological Assessment of Phonxay District, Luang Phrabang Province, Lao PDR. A Minor Field Study.
113. Svensson, S., (2005): Snowcover dynamics and plant phenology extraction using digital camera images and its relation to CO2 fluxes at Stordalen mire, Northern Sweden.
114. Barth, P. von., (2005): Småvatten då och nu. En förändringsstudie av småvatten och deras kväveretentionsförmåga.
115. Areskoug, M., (2005): Planering av dagsutflykter på Island med nätverkanalys
116. Lund, M., (2005): Winter dynamics of the greenhouse gas exchange in a natural bog.
117. Persson, E., (2005): Effect of leaf optical properties on remote sensing of leaf area index in deciduous forest.
118. Mjöfors, K., (2005): How does elevated atmospheric CO2 concentration affect vegetation productivity?
119. Tollebäck, E.,(2005): Modellering av kväveavskiljningen under fyra år i en anlagd våtmark på Lilla Böslid, Halland
120. Isacsson, C., (2005): Empiriska samband mellan fältdata och satellitdata – för olika bokskogområden i södra Sverige.
121. Bergström, D., Malmros, C., (2005): Finding potential sites for small-scale Hydro Power in Uganda: a step to assist the rural electrification by the use of GIS

122. Magnusson, A., (2005): Kartering av skogsskador hos bok och ek i södra Sverige med hjälp av satellitdata.
123. Levallius, J., (2005): Green roofs on municipal buildings in Lund – Modeling potential environmenta benefits.
124. Florén, K., Olsson, M., (2006): Glacifluviala avlagrings- och erosionsformer I sydöstra Skåne – en sedimentologisk och geomorfologisk undersökning.
125. Liljewalch-Fogelmark, K., (2006): Tågbuller i Skåne – befolkningens exponering.
126. Irminger Street, T., (2006): The effects of landscape configuration on species richness and diversity in semi-natural grasslands on Öland – a preliminary study.
127. Karlberg, H., (2006): Vegetationsinventering med rumsligt högupplösande satellitdata – en studie av QuickBirddata för kartläggning av gräsmark och konnektivitet i landskapet.
128 Malmgren, A., (2006): Stormskador. En fjärranalytisk studie av stormen Gudruns skogsskador och dess orsaker.
129 Olofsson, J., (2006): Effects of human land-use on the global carbon cycle during the last 6000 years.
130 Johansson , T., (2006): Uppskattning av nettoprimärproduktionen (NPP) i stormfällen efter stormen Gudrun med hjälp av satellitdata.
131 Eckeskog, M., (2006) Spatial distribution of hydraulic conductivity in the Rio Sucio drainage basin, Nicaragua.
132 Lagerstedt, J., (2006): The effects of managed ruminants grazing on the global carbon cycle and greenhouse gas forcing.