A Survey On Algorithmic Art

5224911: Bastian Boltze
for: Media Art
Steffi Beckhaus, Summerterm 2005, Uni HH

October 24, 2005

Introduction

In summer 2005, the im/ve group[1] has undertaken an explorative excourse into the field of media art. This did not only mean lab research and experimentation - Horst Oberquelle[2] and Steffi Beckhaus[3], in cooperation with Frieder Nake[4], also organized the exhibition "Entgrenzung: Digitale Kunst zwischen Algorithmik und Interaktion"[5], held at the Mensch und Computer 2005[6] conference in Linz. Although historic pieces of algorithmic art, originating in the 1960's and 70's, were the exhibitions primary focus, accompanied by only a few current works, it was nonetheless a contribution to the scene of media art.

Our, the students, introduction to the matter began in context of a seminar led by Steffi Beckhaus, during which we - roughly, of course - reconstructed media art's history on from it's predecessors to the current state of affairs. We were supported not only by one of Germany's pioneers of computer art, Kurt Alsleben[7], giving us the opportunity to participate in his perspective actively involved since the late 1960's, but also by Horst Oberquelle, who has been tracking the legacy of algorithmic art for some time. Since it was announced as a hands-on seminar, we also engaged in the creation of our own home-made pieces of art, some of which were finally included in the aforementioned exhibition.

In the following paper, I will rather chronologically follow the work I have done with respect to my focus in this excourse, namely the foundations and development of algorithmic art. The first section will be a portrayal of certain parts of this line important to my further understanding of the subject. This will, of course, provide only a very narrow view on the events that accompanied the emergence of readily available computers in the field of arts - but since I am

[1]The interactive media / virtual environments group is part of the computer science department at the University of Hamburg. See `http://imve.informatik.uni-hamburg.de`

[2]See `http://asi-www.informatik.uni-hamburg.de/personen/oberquelle/index_d.html`

[3]See `http://imve.informatik.uni-hamburg.de/steffi.htm`

[4]See `http://www.agis.informatik.uni-bremen.de/PERSON/nake.html` or `http://www.medienkunstnetz.de/kuenstler/nake/biografie/forabiography`

[5]See the im/ve group's page about the exhibition at `http://imve.informatik.uni-hamburg.de/ausstellung_mc05.htm`

[6]The conference's homepage can be found at `http://www.mensch-und-computer.de/`

[7]See `http://swiki.hfbk-hamburg.de:8888/NetzkunstWoerterBuch/345`

neither historician nor artist, I think I should not be expected to deliver a scientifically sufficient elaboration. It is, as already said, rather ment as a disclosure of the basis from which I started.

In the second section I will describe the class of regent graphics, one specific class of images that our own experiment bases on. This experiment, the installation “Algorithmik I: Lucy”[8], will be described in the third section; while an explanation of it's implementation details is given in the appendix.

The final evaluation of it's successful conclusion - and the terms it is to be evaluated on - will close this section and the paper.

1 Algorithmic Art

What I here label as algorithmic art is that tradition in the field of visual computer art representing the idea, that it is not primarily - or even not at all - the actual image which is to be considered with respect to the question about the artistic value of some work. Instead it is the formal description or algorithm and the class of images it defines that may or may not have such value and hence be or be not called a piece of art. Of course, there are similar accounts to be found in other fields, like music or literature, but these have undergone a different historic development and won't be considered here any further.[9]

From the area just outlined, it is especially the line emanating from the philosophical work of Max Bense[10] in the 1950's that has been of interest to me; because firstly, it were artists highly influenced by this work, like Georg Nees[11] and Frieder Nake, who provided inspiration for our whole undertaking, and secondly, it has an interesting story to tell with respect to the connection between aesthetics and computer science.

Bense's work was based on certain developments within aesthetic theory in the 1930's which concentrated on the attempt to lay an objectivist basis for the evaluation of aesthetic value, and eventually for some time developed separately from “classic” aesthetic theory. It was, for a most prominent example, George David Birkhoff who tried to define a mathematical measure of aesthetic value. Following this idea, Bense attempted to develop an a account of artistic production, reception and critisism based on cybernetic models of the systems involved: the artist, the recipient as well as the social and economic structures involved in the circulation of art. The persons involved were considered to be fulfilling functions of information processing within a communicative framework, where individual pieces of art did appear as the bearers of a certain kind of information, the aesthetic information. Under the presupposition of sufficient knowledge about the systems involved and their interconnection, this information was assumed to be measurable in information theoretical terms like entropy and redundancy.

[8] See `http://lucy.bspot.de/`

[9] For a more sophisticated elaboration of the whole story, see [Giannetti, Nake 74].

[10] See `http://de.wikipedia.org/wiki/Max_Bense` or `http://www.medienkunstnetz.de/kuenstler/bense/biografie/`

[11] See `http://www.medienkunstnetz.de/kuenstler/nees/biografie/`

Given a set of general rules, the aesthetic information and value of a piece of art was assumed to be determinable in an objectivist way, without need for an actual recipient or critic. And, of course, these rules could on the other hand be used in the production of art in their function as a generative aesthetics. But these rules were not implied by Bense's theory and their exploration was an impetus to the first actual works of algorithmic art.

Today, this whole project of informational aesthetics can - at least on the statements of some of the directly involved - be considered dead. The promise of a way to evaluate or generate pieces of art on a strictly formal account has not fulfilled and is as such of no great further interest, neither to the artists, nor to aesthetic theory in general. It has instead diffused into several other sciences, e.g. the psychology of perception and communcation design.

Nonetheless, the idea that an algorithm can be assigned an aesthetic value with respect to the images it is capable to produce has persisted in a tradition focussing on a different relationship between the artist and his tool than present in more classical forms of painting. Because even in computer art, it is the artist who elaborates the idea of his work - not in terms of colors he manually prepares and applies in certain ways to a canvas, but in more abstract terms describing conditions and aspects he wants to be present in the image; the coincidences introduced through the characteristics of the material are replaced by the pseudo-random signals the algorithm applies to it's calculations.

2 Regent Graphics[12]

In his book "Formel, Farbe, Form" [Nees 95], Georg Nees introduces regent graphics as one formally defined class of images, which offers wide possibilities for experimentation. They are, in principle, an artistic elaboration of Voronoi diagrams, but I will not explain this connection here, because it is not necessary to explain them or understand the algorithm used in their production. Instead, after giving a short formal definition followed by some notes on the algorithm, I will try to sketch the aesthetic possibilities opened by this general framework.

2.1 Formal Definition

A regent graphic is defined over a vector space, which is usually $\mathbb{R}^2$ and thus called the picture plane - although not much speaks agains an extension using spaces of higher dimension. It assigns a color to each point on the plane based on three parameters, i.e.

- a set $Q \subset \mathbb{R}^2$ selected by an arbitrary method, the elements of which are called regents,
- a function $dist : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$, perceived as a distance function, and
- a colorization function col.

[12]Since I could not locate a canonlical translation of the german term "Regentengrafik", I shall translate it like this for the rest of this paper.

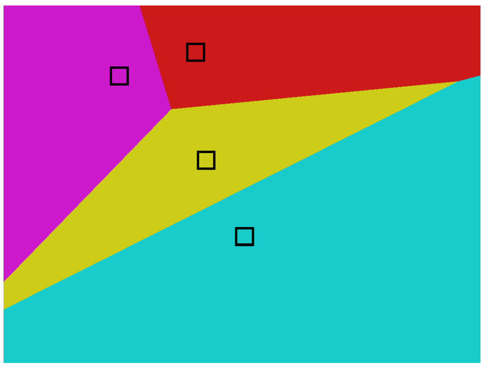


Figure 1: Partitioning of the picture plane by the regents' spheres of influence.

Using the distance function, each point p is related to the nearest regent, which is further identified as his regent q_p:

$$q_p := q \in Q : \nexists q' \in Q : dist\,(p, q') < dist(p, q)$$

Of course, some points are equally nearest to two or more regents - they are just assigned to one of these using an arbitrary method.

This assignment also constitutes the sphere of influence of each regent q as the set of points which have this regent related to them:

$$s_q := \left\{p \in \mathbb{R}^2 \,|\, q_p = q\right\}$$

These spheres are a partitioning of the picture plane, like shown in figure 1 (a), as they are necessarily disjunct and their addition equals the whole plane.

Now, the colorization function, which assigns a color to each point, is usually a function of the values just defined, e.g.

$$color_p = col\,(p, q_p, dist(p, q_p))$$

2.2 Implementation

Unfortunately, no visualization can equal the $\mathbb{R}^2$ with respect to detail - thus, an actual implementation takes samples of the picture plane, producing an image reproduceable on screen or in print. Such an implementation is exemplified by the following pseudocode:

```
 1  BEGIN regentengraphik(canvas, regents)
 2    FOR EACH pixel IN canvas
 3      mydistance = null
 4      myregent = null
 5      FOR EACH regent IN regents
 6        regentdistance = distance(pixel.position, regent.position)
 7        IF ( regentdistance < mydistance OR myregent == null )
 8          mydistance = regentdistance
 9          myregent = regent
10        END IF
11      END FOR
12      pixel.color = color(pixel, myregent, mydistance)
13    END FOR
14  END
```

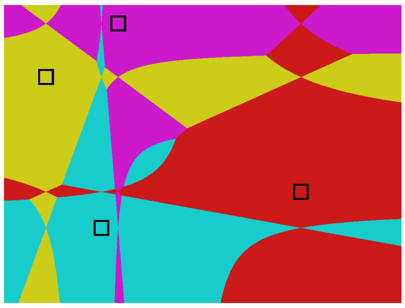
The outer loop (lines 2-13) iterates over all pixels of the image to be produced. In each iteration, the inner loop (lines 5-11) subsequently calculates the distances of the pixel at hand to all of the regents; always carrying on the current lowest distance along with the regent it belongs to. Since the condition in line 7 is not fulfilled if the current minimum distance is equal to the distance calculated for the current regent, a pixel that has the same minimum distance to two or more regents will be associated to the first of them with respect to the ordering among the regents implied by the serial execution of the inner loop. This behaviour may be changed - e.g. by replacing the condition with a less-or-equal test, associating such pixels with the last regent - without harm or significant influence on the picture. Finally, when the regent of minimum distance is known, the pixel is given it's color through the colorization function; based on it's position, the regent and the minimum distance.

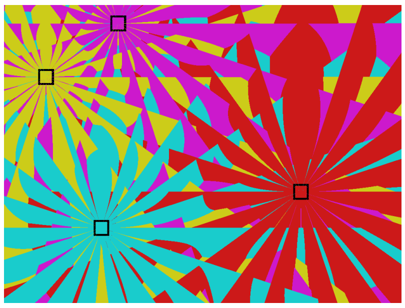
Obviously, this algorithm has a complexity of $O(n * m)$ for the number of pixels n and the number of regents m. There may be more elegant solutions available for the problem than plain brute force, but - as far as I know - these depend on certain properties of the distance function. Since this function is one of the main foci of artistic experimentation, it may not be feasible in most cases to develop an optimized version of the algorithm for each such function in question. And finally, contemporary hardware is capable of executing this generic version quickly enough for a large scale of applications.

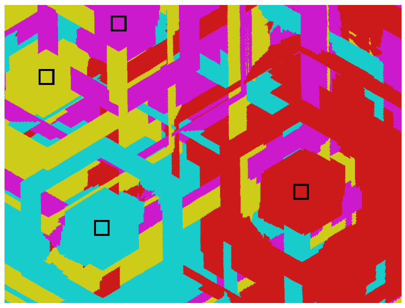
2.3 Aesthetic Possibilities

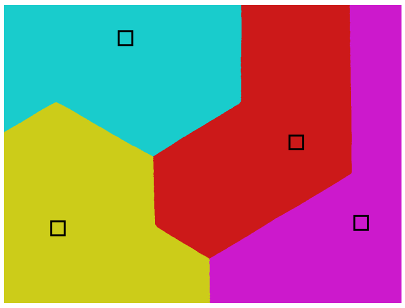
Reviewing the definition and algorithm given above, it is clear that this general concept does not tell much about the appeal of the images resulting from any possible concretion. It is the distance function, the colorization function and the regents positions that do - approximatly in that order with respect to the similarity or dissimilarity between images and with the distance function having the by far highest importance (see Figure 2 for a comparision). I will, therefore, focus on that function and elaborate some of the possibilities for it's choice.

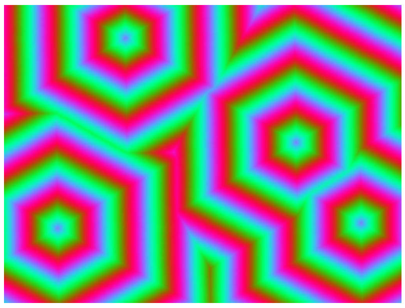
From the visual perspective, a regent graphics' distance function is responsible for shaping the spheres of influence of the regents and stretching the colorization

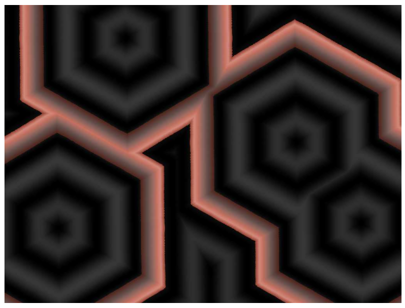

(a)

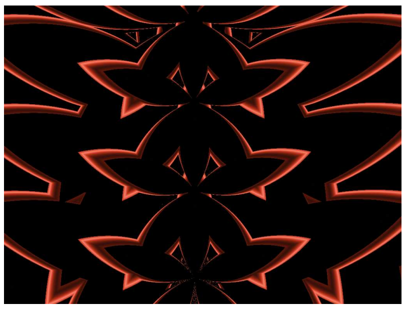

(b)

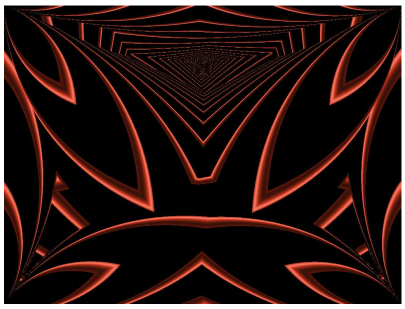

(c)


(d)


(e)


(f)


(g)


(h)

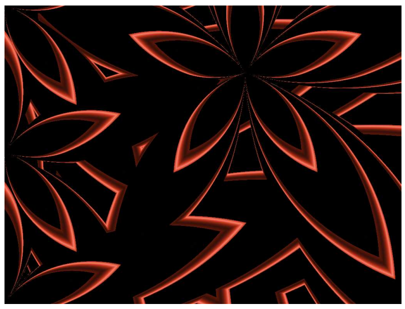

(i)

Figure 2: Influences to the image's appeal on changing only the distance function (a-c), colorization function (d-f), or regents' positions (g-i).

function over these spheres. In general, every function with the required domain and range can do - although pragmatics will most probably pose at least two restrictions: Firstly, it has to be computable, since images are not only to be defined but also actually visualized on some sort of medium. Secondly, it's distribution of values should not be too extreme, because this would - for most colorization functions - produce large empty areas in some parts of the images; and a level of detail too high for any output medium in other areas.

But thinking about a distance measure, one at first naturally remembers the Euclidean distance function. Starting there, other functions may be easily derived: Understanding a circle as the set of points with an equal distance to a center point, functions instantiating very odd shapes as circles can be defined; for example the family of poly-n functions in Figure 2 (d)-(f), which have equilateral polygons as circles.

Still, all these functions create concave, continous and thus well distinguishable spheres of influence for the regents. This can be changed by introducing distance functions that are not metrics in the mathematical sense of the word, i.e. functions that violate one or more of the following conditions:

(I) $d(a,a) = 0$

(II) $d(a,b) = 0 \rightarrow a = b$

(III) $d(a,b) = d(b,a)$

(IV) $d(a,b) \leq d(a,c) + d(c,b)$

For example

$$d(\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle) = abs(x_1 - x_2) \cdot abs(y_1 - y_2)$$

violates at least condition (II), but produces interesting convex influence spheres. See Figure 2 (a)-(c) and (g)-(i) for some impressions.

Additionaly, these functions distort the pattern of the colorization function when stretching it over the image; an effect that can be amplified or applied to simple metric functions by applying an additional trigonometric or potentiation function to the calculated distance.

Going even further, it is not only possible to colorize the spheres of influence in dependence of their regent. The distance function, too, can be parametrized by the regent and/or pixel it is to be calculated for; in effect using different distance functions for different regents and/or areas of the image.

3 Algorithmik I: Lucy

As indicated by it's title, this work by Steffi Beckhaus and myself[13] claims to be an hommage to the period of algorithmic art, or more precisely: the works and

[13]With great help, of course, from Kristopher J. Blom (`http://imve.informatik.uni-hamburg.de/kris.htm`) and Matthias Haringer (`http://imve.informatik.uni-hamburg.de/matthias.htm`).

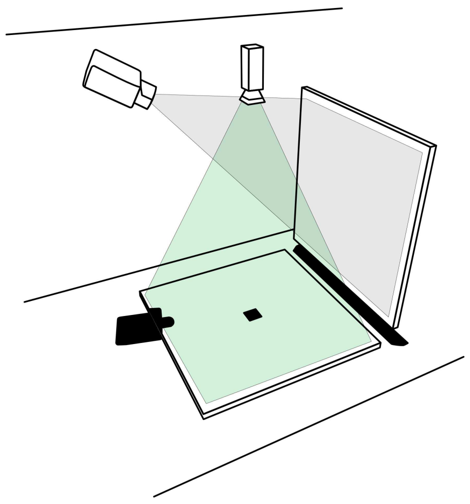


Figure 3: Setup of the installation Lucy

ideas we perceived to constitute that period and which were described already. (Where I will argue later that this is not an unproblematic claim.) It picks up on the image class of regent graphics proposed by Nees, reframing it in an interactive installation based on contemporary hardware, thereby allowing - or better: forcing - the audience to exert an influence on the image.

The setup (see Figure 3) is as follows: Two adjacent surfaces, indicating a cube, mark the space governed by the installation. The horizontal is delimitated on the floor by a carpet of extraordinary color. A ceiling-mounted camera observes this area, while the computer it is attached to detects the positions of any human-sized objects on the carpet. Of course, it transforms these into the regents' positions for a consecutive stream of images it calculates and makes visible on the second, vertical surface by means of a beamer also mounted on the ceiling.

Since the regent graphics algorithm does not produce anything without at least one regent's position specified, this setup presents itself to the approaching observer as some marked area on the floor in front of a lit, but empty projection. It is not until she steps on the carpet that an image appears - and, by virtue of this circumstance and the quality of at least most regent graphics' concretions to expose the points defining them, explains to her the function she fulfills in the production of that image immediately. Subsequently, the rate of approximately 30 fps at which images are produced gives her the impression of a live landscape or organism responding to her movement; an impression that is further amplified by subtle arbitrarities of the images, manifesting in e.g. sudden lightnings or whirling colors. These closed forms, constituted through the other parameters to the algorithm, i.e. the distance and colorization functions, fade from one concretion to the other in short intervals according to an opague internal logic; whereby the spectator's control is limited and the images' predictability reduced.

3.0.1 Results and Conclusion

An experiment ought to be evaluated on behalf of it's success or failure - which most probably also applies to artistic experiments. But what might fit as the criteria of such an evaluation? Of course, the installation did function: the hardware components could be deployed as planed and the software produced the desired results in an acceptable quality without crashing (after some hastily applied fixes, that is). This is, however, just a statement about the technical quality of the implemented system from a developer's perspective, which does not adequately judge the fulfillment of the system's purpose. In fact, the case of an artistic experiment seems to allow for only two categories of evaluation - the aesthetic, and the artistic.

Since there is most unfortunately no informational aesthetic theory to my disposal, that could be applied using formal methods, I can not safely make a statement on the aesthetic value of the images produced - most subjectively, I would describe them as mostly interesting and a little bit flashy over time; but of course my competence in such matters is limited. The field I feel much safer to critisize, the aesthetic value of a software design, comes in two flavors to be dealt with separately with here: On the one hand, the quality a piece of software may or may not possess with respect to elegance, efficiency and cleverness. Although it has many aspects of expendable software in it's implementation of many specific functionalities, the general design of Lucy's software components includes some nice ideas which might be ascribed a certain aesthetic value; and which are at least more elegant than the brute force regent graphics algorithm. But on the other hand, the aesthetic value algorithmic art ascribes certain pieces of software may be much more important, and it is based aesthetic constrains on the produceable images encoded in some algorithm. In this aspect, our implementation exceeds Nees' conception substantialy, for it is not exactly a combination of distance and colorization function we designed - in fact, Lucy itself fulfills this function through the procedure we defined. Unfortunately, this procedure itself is not very elaborated; it does not know anything about the influence the pieces it assembles will have one the output produced. With, for example Harold Cohen's AARON[14], there is an example of a much higher developed work in this area.

Even more complicated is the attempt of an artistic critic of our work, because it presupposes an account on the question of art in general. If, as Frieder Nake states, "art becomes, what is declared as such"[15] [Nake 95?] - and he most probably does not refer to the artist's own exclamation, but to a process within public and specialized spaces - the question if Lucy qualifies as a piece of art can safely be negated. Of course it was shown in an exhibition explicitly declared one of artworks, but in that context it equally explicitly appeared as only a supplemental exhibit to the original works of certain artists. This exhibition had a historic connotation in that it presented unknown works of already known and accepted artists; it did not claim to make a progressive addition to the field of computer arts - and this fact determines the conditions under which the samples presented might enter the processes mention above. In other words: if anything can demand the question of whether it is art or not even be discussed, it has

[14]See `http://www.kurzweilcyberart.com/`

[15]Author's translation of: "Kunst wird, was zu solcher erklärt wird."

itself to take a stand with respect to this question - may it be to demand or reject this label, or disqualify the question as such; which our installation didn't. It did not have the intention to do so.

Still, one might argue that it has an artistic value assured through it's reference to a series of works unquestionably considered art, i.e. those of Georg Nees. But of which nature is this reference? It claims to be an attempt of transgression: Taking up the idea of algorithmik art and reframing it in an interactive context. Unfortunately, this attempt has revealed itself to be awfully amiss. It is exactly the frame this interactivity was given, that disrupts the potential experience of a work of algorithmic art in it's full characterisation, i.e. the final image in conjunction with it's generating algorithm. This form of presentation allows to discover the connection between the formal definition and the aesthetic properties found in the image; in my opinion most preferably by ways of a private interview with the system, examining the effects of variations in the parameters and slight changes on the calculations. But even the solitary, static image pushes the spectator to contemplate about the whereabouts of the characteristics she finds in it. The interactive installation failes in this, as it gives it's audience a fixed frame of possible influence on itself; effectively conceiling it's further conditions of working. Hence, the reconstruction of the work takes place in the image, and not the spectator, denying her the possibility to perceive it in it's full complexity.

Fortunately, a failed experiment does in most cases not mean a set back of the endeavors it was motivated on. In fact, the experiences made in the preliminary studies and the actual realisation have proven to be a rich supplement to the highly technical education in computer sciences; giving insights in one of it's fields of application normaly neither discussed in applied science courses nor interdisciplinary engagements.

References

[Nees 95] Nees, Georg: Formel, Farbe, Form / Computerästhetik für Medien und Design; Springer 1995

[Giannetti] Giannetti, Claudia: Kybernetische Ästhetik und Kommunikation; Medien Kunst Netz; available online at `http://www.medienkunstnetz.de/themen/aesthetik_des_digitalen/kybernetische_aesthetik/`

[Nake 74] Nake, Frieder: Ästhetik als Informationsverarbeitung; Springer 1974

[Nake 95?] Nake, Frieder: Einmaliges und Beliebiges / Künstliche Kunst im Strom der Zeit; available online at`http://userpage.fu-berlin.de/~zosch/werkstatt/nake.html`; date and printed version not trackable

A Implementation details

On the part of software, the installation consists of two applications: rcam, which is responsible for object detection, and mmarts, which synthesises images implementing the regent graphics' algorithm. Both are written in C++ and utilitze hardware accelerated frament shader programs via the OpenGL library.

In the setup used at the Mensch und Computer 2005, the availability of those was provided by a video card with NVidia GeForce 6600 GT chipset connected to the beamer. Input came from a Sony DV cam connected via FireWire. On this hardware, approximate framerates of 25 fps for input processing and 30 fps for the resulting image stream were achieveable.

A.1 rcam - Object Detection

A.1.1 Input

The application expects it's input to be a sequence of 720 by 576 wide 3-byte RGB frames, which is read from standard input. It is rather ignorant against the framerate, at which images are received: It will not block, if no or only a partial frame is available at a given moment, and will silently drop superflous frames.

To allow direct streaming from an attached camera, which is the intended mode of operation, a 2.0 version of the dvgrab utility[16] has been patched to provide output in this format.

A.1.2 Image Preprocessing

Before object recognition takes place, the program determines the relevant areas of the processed image, i.e. the parts that are significantly different in comparision to the reference image. At first, the image, as well as the reference image, is scaled down by a variable factor (of 2, by default) and converted to 3-byte HSV space. Then, a difference image is constructed through the absolute values of the componentwise subtraction of the two images.

A set of conditions evaluated on each pixel of the difference image determines, if this pixel has changed enough to be considered relevant. Currently, this is supposed to be the case, if

- the hue difference is greater than 10 percent, or
- the saturation difference is greater than 15 percent, or
- the value difference is greater than 30 percent.

[16]dvgrab is a utility to capture data from a DV camera. It is open source software available under the GLP. See http://www.kinodv.org/.

While the scaling is carried out by the CPU, the further steps are implemented by a fragment shader program running on the system's GPU. The main application utilizes OpenGL to transfers the images as textures and render a rectangle projected screen-filling on a window of the images size. When the shader program has finished, the window contains a picture representing the relevance values of the individual pixels in it's red channel as either 0 or 1. This difference map is returned to system memory and, in conjunction with the original image, processed by the object detection stage.

A.1.3 Object Recognition

In general, blobs are found using the following procedure: Starting with one pixel known to differ from the reference image, all direct neighbours of the blobs pixels are checked and added to the blob if they also differ from the reference image. Finally, if the blob excedes a certain size measured in pixels, the coordinates of all it's pixels are summed up and divided by their number to determine the blob's center.

For each frame, firstly the blobs present in the last frame are checked for their persistence. The algorithm starts with the pixel at the blob's last known center coordinate. Moreover, this is carried out for all blobs in parallel, each blob having the chance to check one neighbour pixel before it is next blob's turn - if they where checked sequentially, one blob could absorb another one if they were overlapping in the current camera image.

Afterwards, each pixel differing from the reference image in an area at the images borders 10 percent the size of image width and height respectively is made the starting point for the blob detection algorithm to detect people entering the area observed by the camera. These searches are carried out in sequence to safely identify the largest continous areas as one blob.

Of course, this algorithm has it's flaws: Firstly, it may occur that it looses track of people or objects in it's area - e.g. if they move so fast that the point determined as their visual center in one frame does not lie within the area covered by them in the next frame, or if two people entered the camera's visual field conjoinedly and were identified as one blob. In both cases, the fact that new blobs are detected only in the border regions forces these persons to leave and reenter the area to be tracked again. On the other hand, this scheme prevents a lot of superflous blobs, which appear if a person, for only one frame, does not constitute a continous area in the camera image, because parts of her clothes can not be distinguished from the floor. This leads to the second problem, which appears when someone enters the visual field very slowly. It may happen then, that different parts of her body, e.g. an arm and a leg, are already visible without seeming connected by a body and hence are identified as different blobs, which the person in question will carry around as long as she stays in view.

A.1.4 Output

The programs output consists of a list of normalized blobs' center coordinates identified by a blob number that stays fixed for each blob; together with an indication of the amount of time since the blob was firstly and lastly detected, which allows the rendering application to visualize the appeareance and disappeareance of blobs smoothly.

Output can be send in binary form through a TCP connection or to some designated host by UDP; additionaly it is presented in human-readable form on standard output, which may of course be redirected. In general, output is handled by an instance of a class extending Sender; hence other output methods can easily be implemented.

A.2 mmarts - Picture Synthesis

The mmarts application synthesis a continous stream of images in an infinite loop. The picture is represented on screen by a set of polygons rendered using OpenGL, which are mapped to parts of the picture plane specified by their texture coordinates. In the easiest case, just one rectangle with texture coordinates from $< 0, 0 >$ to $< 1, 0.75 >$, representing this section of the picture plane, is rendered, preferably filling the whole screen. A fragment shader program implementing the picture function is assigned to these polygons, so each pixel belonging to one of them is in fact a sample of the picture function at the respective texture coordinates. The composition of these shader programs is the most oustanding functionality implemented in the application.

A.2.1 Input / Control

In normal operation, the application receives as it's only input the regents' coordinates from the rcam application by means of a TCP connection or single UDP packets. In symmetry to this application, input is handled by an instance of a class extending Receiver, allowing for other input methods to be added.

In conjuction with a mechanism to control mutators' (see below) behaviour, the DummyReceiver class, which allows manual placement of regents, images can be directly composed by mouse and keyboard.

Of course, the application provides some hotkeys to control it's operation, e.g. switching between fullscreen and windowed mode, quitting, etc.

A.2.2 Shader Synthesis

As noted before, the main program randomly assembles new frament shader programs every few minutes. More precisely, this functionality is implemented by the Lab class and triggered by an instance of ShaderControl.

A set of code fragments, called molecules, in (an extended version of) the Cg shader language provides the base material for this synthesis. They are located in the applications data directory and loaded on startup. Indicated to the application through a magic comment in the first rows, each molecule implements exactly one of the following functionalities:

- A basic control function defining the general algorithm,

- a distance function, or
- a colorization function.

In fact, colorization molecules are further differentiated into

- color modification molecules, which apply some filtering function to the output of another color molecule,
- blending molecules, which are capable of rendering a stable pattern independent of the regent's positions when indicated to do so by a special shader parameter, used to mask a freeze introduced when a shader program is compiled, and
- basic colorization molecules not refering to other molecules.

The arbitrary combination of molecules is made possible by that they may contain unterspecified calls to other molecules, they may just specify the called molecules implemented functionality. A colorization molecule may, for example, call another colorization molecule and apply a filter to it's result, but without specifing exactly which one. Specifically a base molecule will most probably contain several calls to an unspecified distance function and one, maybe more, calls to also arbitrary colorization functions.

Synthesis of a complete picture function now proceeds straightforward by selecting one base molecule and recursively choosing random molecules of the respective types to fill the underspecified calls. The only restriction to these choices ensures that the first two colorization molecules called from the base molecule are blending molecules; one to blend into the newly created shader, matching the one used to blend out the last active shader, and one randomly choosen to blend out the shader when it's job will be done.

Maybe the greatest difficulty in this procedure concerns the parameters needed by various molecules: On the one hand, a molecule may need a specific parameter, like the texture coordinate of the fragment processed, or even one introduced by itself, but it may be come to be called by a molecule that does not know about this specific parameter. On the other hand, a molecule may want to call another molecule with a forged value for some parameter, i.e. to distort patterns based on distance, but can't be sure the called molecule even recognizes this parameter.

To resolve this problem, all data exchange between molecules is defined using semantic markers. A molecule specifies a semantic binding for each parameter it's main function takes. Furthermore, each value to be passed over in a molecule call is also marked by such a binding. When filling the call slots of a molecule in synthesis, the program examines the parameter lists of call and called module and proceeds as follows:

- If a parameter (identified by it's semantic binding) is present in both lists, the expression specified in the call will be passed to called module.

- If it is specified in the call but not needed by the called module, it is dropped from the call.
- If it is needed by the called module but not specified in the call, it is first checked if it is already on the calling module's parameter list; and if not, it is added to this list using a new variable name. Now that it is guarranteed to be present in the calling module under some variable name, this variable is passed in the call. (Note: This is error prone: First, the variable bound to the parameter may be modified by the calling module before the call in question, and second, the calling module may consist of more than one function, so that when the parameter is introduced through this procedure, it is present in it's main entry function, but not in the other function that may contain the call.)

Predefined semantic markers exist to bind variables to the parameters introduced to the shader by the main program: The number of regents present, their positions and presence values, the current state of blending into or out of the shader, etc. In addition, molecules may define variable parameters with specific properties, as explained in the following sections.

A.2.3 Texture Parameters

Using a special syntax for the semantic marker of a parameter in it's parameter list, a molecule may request a texture to be loaded and a sampler2D on that texture to be bound to the parameter. On synthesis, these markers are parsed, textures loaded and bound to a free texture unit. The parameter will then be propagated to the function list of the shader program's main function (as described above) and bound to the selected texture unit using the Cg languages own syntax for semantic binding.

A.2.4 Realtime Picture Manipulation

Another special syntax for semantic markers allows a molecule to indicate that a parameter value may be modified within defined boundaries while the shader is running to change the results of the implemented function and thus the appeal of the generated picture. More specifically, the following kinds of these mutable parameters can be specified:

- Floating point parameters, mutating between a lower and upper bound in intervals randomly choosen between a minimum and maximum time of stability; the transition between old and new value proceeding by linear interpolation over an also randomly choosen timespan.
- RGB color parameters (implemented thru the Cg-languages type float3), mutating between upper and lower bounds for each color component just like the floating point kind.
- Frame parameters, repeatedly running from 0.0 to 1.0 within a specified number of milliseconds.

On synthesis, the program parses these specifications and creates an instance of the respective class extending Mutator, which is bound to the parameter and provides the requested behavior at runtime.