Manovich, Lev. "Soft evolution." *Software Takes Command*. New York: Bloomsbury, 2013. 199–240. *Bloomsbury Collections*. Web. 26 Apr. 2023. <<http://dx.doi.org/10.5040/9781472544988.ch-004>>.

Downloaded from Bloomsbury Collections, www.bloomsburycollections.com, 26 April 2023, 14:41 UTC.

Copyright © Lev Manovich 2013. You may share this work for non-commercial purposes only, provided you give attribution to the copyright holder and the publisher, and provide a link to the Creative Commons licence.

CHAPTER FOUR

Soft evolution

Algorithms and data structures

What makes possible the hybridization of media creation, editing, and navigation techniques? To start answering this question we need to ask once again what it means to simulate physical media in software. For example, what does it mean to simulate photography or print media?

A naïve answer is that computers simulate the actual media objects themselves. For example, a digital photograph simulates an analog photograph printed on paper; a digital illustration simulates an illustration drawn on paper; and digital video simulates analog video recorded on videotape. But that is not how things actually work.

What software simulates are the physical, mechanical, or electronic *techniques used to navigate, create, edit, and interact with media data*. (And, of course, software also extends and augments them, as discussed in detail in Part 1.) For example, the simulation of print includes the techniques for writing and editing text (copy, cut, paste, insert); the techniques for modifying the appearance of this text (change fonts or text color) and the layout of the document (define margins, insert page numbers, etc.); and the techniques for viewing the final document (go to the next page, view multiple pages, zoom, make bookmark). Similarly, software simulation of cinema includes all the techniques of cinematography such as user-defined focus, the grammar of camera movements

(pan, dolly, zoom), the particular lens that defines what part of a virtual scene the camera will see, etc. The simulation of analog video includes a set of navigation commands: play forward, play in reverse, fast forward, loop, etc. In short: *to simulate a medium in software means to simulate its tools and interfaces, rather than its "material."*

Before their softwarization, the techniques available in a particular medium were part of its "hardware." This hardware included instruments for inscribing information on some material, modifying this information, and—if the information was not directly accessible to human senses such as in the case of sound recording—presenting it. Together the material and the instruments determined what a given medium could do.

For example, the techniques available for writing were determined by the properties of paper and writing instruments, such as a fountain pen or a typewriter. (The paper allows making marks on top of other marks, the marks can be erased if one uses pencil but not pen, etc.) The techniques of filmmaking were similarly determined by the properties of film stock and the recording instrument (i.e. a film camera). Because each medium used its own distinct materials and physical, mechanical, or electronic instruments, each also developed its own set of techniques, with little overlap.

Thus, because media techniques were part of specific incompatible hardware, their hybridization was prevented. For instance, you could white out a word while typing on a typewriter and type over—but you could not do this with the already exposed film. Or, you could zoom out while filming progressively revealing more information—but you could not do the same while reading a book (i.e. you could not instantly reformat the book to see a whole chapter at once.) A printed book interface only allowed you to access information at a constant level of detail—whatever would fit a two-page spread.¹

Software simulation liberates media creation and interaction techniques from their respective hardware. The techniques are translated into software, i.e. each becomes a separate algorithm.

¹ This was one of the conventions of books which early twentieth-century book experiments by modernist poets, and designers such as Marinetti, Rozanova, Kruchenykh, Lissitzky, and others worked against.

And what about the physical materials of different mediums? It may seem that in the process of simulation they are eliminated. Instead, media algorithms, like all software, work on a single material—digital data, i.e. numbers.

However, the reality is more complex and more interesting. The differences between materials of distinct media do not simply disappear into thin air. Instead of a variety of physical materials computational mediums use different ways of coding and storing information—different data structures. And here comes the crucial point. *In place of a large number of physical materials, software simulations use a smaller number of data structures.*

(A note on my use of the term “data structure.” In computer science, a data structure is defined as “a particular way of storing and organizing data in a computer so that it can be used efficiently.”² The examples of data structures are arrays, lists, and trees. I am going to appropriate this term and use it somewhat differently, to refer to “higher-level” representations which are central to contemporary computational media: a bitmap image, a vector image, a polygonal 3D model, NURBS models, a text file, HTML, XML and a few others. Although the IT, media, and culture industries revolve around these formats, they do not have a standard name of their own. For me the term “representation” is too culturally loaded while the term “data type” sounds strictly technical. I prefer “data structure” because it simultaneously has a specific meaning in computer science and also a meaning in humanities—i.e. the “structure” part. The term will keep reminding us that what we experience as “media,” “content” or “cultural artifact” is technically a set of data organized in a particular way.)

Consider all different types of materials that can be used to create 2D images, from stone, parchment and canvas to all the dozens types of paper, that one can find today in an art supply store. Add to those all of the different kinds of photographic film, X-Ray film, film stocks, celluloid used for animation, etc. Digital imaging substitutes all these different materials by employing just two data structures. The first is the bitmapped image—a grid of discrete “picture elements” (i.e., pixels) each having its own color

² Paul E. Black, ed., entry for “data structure” in *Dictionary of Algorithms and Data Structures*, U.S. National Institute of Standards and Technology, <http://xlinux.nist.gov/dads/>

or gray-scale value. The second is the vector image, consisting of lines and shapes defined by mathematical equations.

So what then happens to all the different effects that these physical materials were making possible? Drawing on rough paper produces different effects from drawing on smooth paper; carving an image on wood is different from etching the same drawing in metal. With softwarization, all these effects are moved from “hardware” (physical materials and tools) into software.

All algorithms for creating and editing continuous-tone images work on the same data structure—a grid of pixels. And while they use different computational steps, the end result of these computations is always the same—a modification in the colors of some of the pixels. Depending on which pixels are being modified and in what fashion, the algorithms can visually simulate the effects of drawing on smooth or rough paper, using oils on canvas, carving on wood, and making paintings and drawings using a variety of physical instruments and materials.

If particular medium effects were previously the result of the interaction between the properties of the tools and the properties of the material, now they are the result of different algorithms modifying a single data structure. So we can first apply the algorithm that acts as a brush on canvas, then an algorithm that creates an effect of a watercolor brush on rough paper, then a fine pen on a smooth paper, and so on. In short, the techniques of previously separate mediums can now be easily combined within a single image. And since media applications such as Photoshop offer dozens of these algorithms (presented to the user as tools and filters with controls and options), this theoretical possibility becomes a standard practice. The result is a new hybrid medium that combines the possibilities of many once-separate mediums.

Instead of numerous separate materials and instruments, we can now use a single software application whose tools and filters can simulate different media creation and modification techniques. The effects that previously could not be combined since they were tied to unique materials are now available from a single pull-down menu. And when somebody invents a new algorithm, or a new version of already existing algorithm, it can easily be added to this menu using the plug-in architecture that became standardized in the 1990s (the term “plug-in” was coined in 1987 by the developers of

Digital Darkroom, a photo editing application³). And, of course, numerous other image creation and modification techniques that did not exist previously can be also added: image arithmetic, algorithmic texture generation (such as Photoshop's Render Clouds filter), a variety of blur filters, and so on (Photoshop's menus provide many more examples).

To summarize this analysis: software simulation substitutes a variety of distinct materials and the tools used to inscribe information (i.e., make marks) on these materials with a new hybrid medium defined by a common data structure. Because of this common structure, multiple techniques that were previously unique to different media can now be used together. At the same time, new previously non-existent techniques can be added as well, so long as they can operate on the same data structure.

(Note: many standard contemporary image formats including the Photoshop .psd format are much more complex than a simple pixel grid—they can include alpha channels, multiple layers, and color profiles; they can also combine bitmapped and vector representations. However, in this discussion, I am only talking about their common denominator—what the algorithms work on when an image is loaded in memory—an array of pixels holding color values.)

Let us now look at another example of what happens with physical materials of different mediums when they are simulated in software. Consider 3D modeling software such as Blender, Maya, 3ds Max Studio, LightWave 3D, or Google's SketchUp. These applications provide the techniques for defining 3D forms which were previously "hardwired" to different physical media. For example, you can use sculpting tools to create a rounded form as though you are using clay. 3D applications also provide dozens of new techniques for defining and modifying forms not available previously: bevel, extrude, spherize, randomize, boolean operations, smooth, loft, morph, simplify, subdivide, and so on.⁴ As with image editing software, new techniques can always be added as long as they operate on the standard data structures already used by 3D software. (The most common ones are polygonal models

³ http://en.wikipedia.org/wiki/Digital_Darkroom (February 19, 2012)

⁴ A list of a subset of the operations that work on 3D models made from polygons is provided in http://en.wikipedia.org/wiki/Polygon_modeling#Operations

and NURBS models.⁵ The former consist of flat polygons; the latter are defined by smooth curves.) These data structures are the new “materials” that software substitutes for a variety of physical materials used by humans to create physical 3D forms such as stone, wood, clay, or concrete.

These two examples of raster image and 3D model data structures should make clear why it is incorrect to think that computers always work on a single digital material, so to speak, i.e. the binary code made from 0 and 1. Of course this is what happens on a low-level machine level—but this is largely irrelevant as far as application software users and people who write this software are concerned. Contemporary media software contains its own “materials”: data structures used to represent still and moving images, 3D forms, volumes and spaces, texts, sound compositions, print designs, web pages, and other “cultural data.” These data structures do not correspond to physical materials in a 1:1 fashion. Instead, a number of physical materials are mapped onto a single structure—for instance, different imaging materials such as paper, canvas, photographic film, and videotape become a single data structure (i.e., a bitmapped image). This *many to one mapping from physical materials to data structures is one of the conditions which enables hybridization of media techniques.*

What is a “medium”?

I have spent considerable time analyzing the specificity of media software in relation to pre-digital media. This analysis allows us to better understand why hybridity became the next stage in computer metamedium evolution and to begin tracing the mechanisms of hybridization. Let us now see if we can use our new insights to answer one of the key questions of this book: what is “media” after its softwarization?

To avoid confusion: I am not talking about the actual *content* of media programs and sources, be they television programs, newspapers, blogs, or the terrain in Google Earth. There are already a number of academic disciplines which study

⁵ http://en.wikipedia.org/wiki/3D_modeling

media content and its reception: Media Studies, Communication, Journalism, Film and TV Studies, Game Studies, Cultural Studies, and Internet Studies. I am also not going to talk about *media industries*—production, distribution, reception, markets, economic aspects, etc., because academic disciplines already analyze these extensively. However, they usually do not so closely analyze the tools, technologies and workflows used to produce media content. Even when they do this analysis, it is only done in relation to the tools of a particular media field. This is because the modern academic study of culture follows the commercial culture industries' strict division by type of content. Thus, Game Studies looks at games, Film and TV Studies looks at films and television programs, Design Studies looks at design, Internet Studies looks at the web, etc. Because of these divisions, these disciplines ignore the common features of all media and cultural production being done today which are the result of their reliance on the same technology—*application software for media authoring and editing*. (This is one of the reasons why we need a Software Studies perspective—to focus our attention on common cultural patterns related to the use of software technology in all of these diverse cultural fields and media industries.)

I will also bracket media reception—in other words, I will try to define what “media” is today for its creators as opposed to media consumers. (While in the 2000s there were many discussions about the blurring of these definitions because of the falling prices of tools and emergence of social media, in practice they did not get erased.) And finally, people who know computer programming and can create media by writing programs will also have a different understanding of media—but the majority of content creators use only application software.

The universe of the users of application software includes “creative professionals”: motion graphics artists, graphic designers, photographers, video editors, product designers, architects, visual artists, etc. It also includes “prosumers” (or “pro-ams”) making anime remixes, editing documentary videos which they will upload to YouTube or Vimeo, shooting photos which they will post to their Pro Flickr accounts, or uploading their art images to deviantArt.

I want to understand what it means to create “media” for all these people as defined by the possibilities of the software they are using—Photoshop, Gimp, Illustrator, InDesign, After Effects,

Final Cut, Premiere, CinePaint, Maya, Dreamweaver, WordPress, Blogger, Flash, OpenOffice, Pages, Microsoft Word, Flame, Maya, and so on. (And, speaking of Word and other text processing applications, I should also add millions of people who use it daily and who, therefore, can be considered experts or at least prosumers in at least one medium—that of text authoring and editing.)

Recall one of the dictionary definitions of a “medium” which opens *Understanding Metamedia* (Chapter 2): “A specific kind of artistic technique or means of expression as determined by the materials used or the creative methods involved.” (For example, “the medium of lithography.”) Thus, different mediums have different techniques, means of expression and creative methods. These differences certainly do not disappear when we switch to software applications. For example, besides the obvious representational and expressive difference between 3D models and moving images, a designer who is modeling a game character in Maya and a designer who is making animation in After Effects will have access to different sets of tools. But are there some conceptual similarities between the way these two designers will be working because they both use media software?

In short: *what is “media” today as defined by software applications used to create and edit it?*

As I have already discussed, earlier physical, mechanical, and electronic media consisted of two components: materials used to hold information and the tools or equipment used to record, edit, and view this information. For example, the “film medium” used film stock for information storage, a film camera for recording, a projector for showing films, and editing devices such as Moviola and Steenbeck. The medium of hand engraving used metal plates (typically copper) to hold information, and special, hardened steel tools to create it by making grooves in the plate.

Do these two components find their analogs in software? Here is one answer which we can give to this question: *Materials become data structures; the physical, mechanical, and electronic tools are transformed into software tools which operate on these data structures.* From this perspective, regardless of the particular media field, all designers and artists working with media software are doing the same thing: using the tools provided by the software to create, modify, and edit data organized in particular data structures.

This answer is compelling but not precise. As I already discussed, many materials are mapped into a single data structure. Thus, the move from physical media to software apps involves a redistribution of the roles previously played by the physical tools and materials. When I use a watercolor brush and a rough-textured paper, the resulting brushstrokes are equally the result of the brush, the liquid, and the paper. But when I use a “watercolor” brush in Photoshop, or apply a “watercolor” filter to an already existing image, the result is determined solely by an algorithm, that modifies the colors of the pixels in a particular manner. The pixels are only memory locations, which hold the color values—they do not have any properties of their own, unlike physical materials.

Therefore, we do not have a one-to-one mapping between physical materials and data structures. The same data structure (such as a bitmap image) can be used to simulate many imaging techniques: from watercolor and engraving to photography. To make this concrete, look at all JPEG images on your computer. Some are your photographs uploaded from a mobile phone or digital camera; others are small graphical icons used by various applications; still others may be the notes you made with a note-taking app, and so on. The same data structures hold multiple media.

This is why rather than stating that materials turn into data structures while tools turn into algorithms, it would be more correct to say that *a medium as simulated in software is a combination of a data structure and set of algorithms*. The same data structure can be shared across multiple medium simulations, but at least some of the algorithms will be unique to each medium.

We have arrived at a definition of a software “medium,” which can be written in this way:

$$\text{Medium} = \text{algorithms} + \text{a data structure}$$

Algorithms and data structures happen to be two fundamental elements of computer programming. In fact, one of the most influential books in the history of computer science is Niklaus Wirth’s *Algorithms Plus Data Structure Equals Programs* published in 1975. Wirth and other computer scientists conceptualized the intellectual work of programming as consisting of two interconnected parts: creating the data structures which logically fit with the task which needs to be done and are computationally

efficient, and defining the algorithms which operate on these data structures.

We can use this conceptual model of computer programming to refine our understanding of what media applications do. All applications, including media software, are computer programs, so internally they involve algorithms working on data structures. This point by itself is not very revealing. What is more important is that these two elements, in my view, also define a mental model a user has of the application—i.e., how the user understands what an application presents to him/her, and what s/he is doing while using that application. In other words, the user's mental model reflects the abstract structure of a computer program (algorithms operating on data structure) that drives the particular media application software.

Inside the application environment, a user is working with one or more documents which contain content structured in a particular way. The user is aware of the importance of this structure—even though media applications do not use the term “data structure.” The user understands that the data structure determines what content can be created and the operations which can be used to shape and modify it. If I choose vector graphics as my format, this implies that I will be creating straight and curved lines and gradients; I will be able to reshape any line in the future without losing quality; I will also be able to get perfect print output at any resolution. If I select bitmap image as my format, I can work with photographs and other continuous-tone images, blurring and sharpening details, painting over, applying filters and so on; but the price for this flexibility is that the image will exhibit undesirable artifacts if I enlarge it many times.⁶ (In practice, this selection is done when a user chooses the primary application to assemble the project. Choosing Illustrator implies that you will be working with vector graphics; choosing Photoshop implies that you will be working with bitmap images. Although each program also supports working with the opposite image type, the majority of its functions and its interfaces are organized around its “native” type.)

⁶ An example of a popular vector file format is Illustrator's AI format; JPEG and PSD are examples of bitmap file formats. Certain file formats such as EPS, PDF, and SWS can hold both vector graphics and bitmap images.

One more reminder because this is important: the term “data structures” has a particular meaning in computer science, referring to how data to be processed by a program is organized. As I have already said, while I want to retain the core idea of data organization, I am not interested here in how this actually happens on the low-level (i.e., whether the program organizes data using arrays, linked lists, etc.). Instead, I am using this term to refer to the *level of data organization, which is made visible and accessible to a user* and thus becomes a part of his/her mental model of the media creation and editing process. (For instance, when I work with bitmap images in Photoshop, I can zoom to examine individual pixels; I can check resolution of my image in pixels, I can choose the diameter of my brush, again in pixels; and so on. All this reminds me that my image is a pixel grid.)

To summarize this discussion: I suggested that both theoretically and also experientially—at least for the users who have more than casual experiences with media applications—“media” translates into two parts which work together. One part is a small number of basic data structures (or “formats”) which are the foundation of all modern media software: bitmap image, vector image, 3D polygonal model, 3D NURBS model, ASCII text, HTML, XML, sound and video formats, KML, etc. The second part is the algorithms (we can also call them “operations,” “tools” or “commands”) that operate on these formats.

The ways in which these two parts are actualized in media applications require additional discussion. First, different applications often add more details on top on these basic types to give them additional functionality. For instance, an image as it is defined by capabilities of Photoshop (a professional and more expensive application) is substantially different from the way it is defined in the much less expensive Apple’s iPhoto or Google’s Picasa. As defined by Photoshop, an image is a complex hierarchical structure. (This description refers to Adobe Photoshop CS5.5.) At its base is the basic bitmap image: a grid of pixels. A pixel is a minimal element that a user can select and modify. This is a basic data structure Photoshop shares with all other image editors. (Note that while the user cannot select parts of pixels, the actual algorithms often work on a sub-pixel level.) A document in Photoshop can contain many such pixel grids; they are referred to as “layers.” The layers can be collected in groups. They can also form “layer comps”—alternative

versions of the composition layout. Any single document layer can also have many states: a user can make it visible or invisible, change its transparency, the way it interacts with layers underneath, etc. Photoshop also provides special adjustment layers that do not contain any pixel content; according to the program documentation, “An adjustment layer applies color and tonal adjustments to your image without permanently changing pixel values.”⁷ Additionally, an image can also have a number of masks that define which areas of an image can be edited. Other elements that Photoshop adds to the basic image structure include paths and vector-based graphics and types.

Photoshop’s UI uses a number of windows and menus to present this complex image structure, with all its possibilities. The Document window displays the actual composition. The Layers panel shows all layers, layer effects, and layer groups in the composition. The Channels panel shows the color components of an image (such as R, G, B). Each of these windows has a number of controls. Additionally, various menu items are dedicated to creating, viewing, and modifying all the possible image parts.

If data structures form one part of a user’s mental model of media creation in software applications, the operations that can be used on these structures comprise the second part. That is, the user also understands that the process of defining and editing the content involves the sequential application of different operations provided by an application. Each operation corresponds to an algorithm which either performs some actions on the already existing data or generates new data. (Photoshop’s Wave filter is an example of the former, while the Render Clouds filter is an example of the latter.)

In contemporary media software, the tools that are represented by items in menus are often referred to as “commands.” The term “tool” is reserved for those frequently used operations that are given their own icons and can be selected directly without having to navigate through the menus. (I will use the word “tool” to refer to both types.) The applications group these related operations together. For example, Photoshop CS5 collects its key tools in the Tools Panel; additional tools are found under top-down menus

⁷ http://help.adobe.com/en_US/Photoshop/11.0/

named Edit, Image, Layer, Select, Filter, and View. Many media applications also make available additional tools in the form of scripts that can be run from within the application. For example, in addition to dozens of commands already available in Photoshop CS5's menus and panels a user can also run Adobe or third-party scripts which appear under File > Scripts. These scripts can be written in JavaScript, VBScript, AppleScript, and other scripting languages. Finally, people who use command-line interfaces such as Unix (or Linux) can also use a third type of operations—separate software programs which are run directly from a command line. For example, two very widely used programs for image and video conversion and editing are ImageMagic and FFmpeg.⁸ Since these types of programs do not have a GUI interface, they are not suitable for interactive image editing; however, they excel in automation and are often used to perform batch operations (such as conversion from one file format to another) on a large number of files at once.

Regardless of whether the media tools are presented via GUI, as scripts, or as separate programs available from a command line, they all have one thing in common—they can only work on particular data structures. For instance, image-editing applications define dozens of tools for editing bitmap images⁹—but these tools would not work on vector graphics. In another example, the techniques for modification of 3D models that define the volume of an object are different from the techniques that can operate on 3D models that represent the object's boundary (such as polygonal models).

To make an analogy with language, we can compare data structures to nouns and algorithms to verbs. To make an analogy with logic, we can compare them to subjects and predicates. Like all metaphors, these two highlight and distort, reveal and hide. However, I hope that they can help me to communicate my key point—the dual nature of a “medium” as defined by software applications.

We have now arrived at one possible answer to the question we posed in this book's introduction: what is media today as defined by software applications for its creation and editing? *As defined by application software and experienced by users, a “medium”*

⁸ <http://www.imagemagick.org/>, <http://ffmpeg.org/>

⁹ This Wikipedia article lists images editing operations common to these programs: http://en.wikipedia.org/wiki/Image_editing

is a pairing of a particular data structure and the algorithms for creation, editing and viewing the content stored in that structure.

Now that we have established that computational media involves combining algorithms and data structures, we can also better understand the distinction between media-specific and media-independent techniques that I made earlier. *A media-specific technique is an algorithm that can only operate on a particular data structure.* For example, blur and sharpen filters can only work on bitmapped images; the “extrude” operation commonly used in 3D programs to make 3D models can only be applied to a vector curve. In contrast, *a media-independent technique is a set of algorithms that all perform a conceptually similar task but are implemented to work on a number of data structures.* I mentioned examples of these techniques when I introduced this concept—sort, search, zoom, cut, copy, and paste, randomize, various file manipulations (copy, email, upload, compress, etc.), etc.

To explain how media-independent techniques can be implemented, let us look at the Copy, Cut and Paste commands. These operations already existed in some computer text editors in the 1960s. In 1974–1975 Larry Tesler implemented these commands in a text editor as part of Xerox PARC’s work on a personal computer.¹⁰ Recognizing that these commands can be used in all types of applications, the designers of Xerox Star (released in 1981) put dedicated keys for these commands in a special keypad.¹¹ The keypad contained keys marked Again, Find, Same, Open, Delete, Copy, Merge, and Move. A user could select any object in an application or on the desktop and then select one of these commands. Xerox PARC team called them “universal commands.” Apple similarly made these commands available in all applications running under its unified GUI but got rid of the dedicated keys.¹² Instead, the commands were placed under the Edit pull-down menu.

The idea that a user can select objects in any document regardless of the media, or any file, and use the same set of commands on

¹⁰ http://en.wikipedia.org/wiki/Cut,_copy,_and_paste

¹¹ http://en.wikipedia.org/wiki/Xerox_Star (February 20, 2012)

¹² For a close-up view of the dedicated keypad for universal commands and demonstration of their operations, see the part showing the Xerox Star keyboard in this video: <http://www.youtube.com/watch?v=Cn4vC80Pv6Q&feature=relmfu> (August 4, 2012).

these objects is among the most important inventions of the Xerox PARC team. It gives the user a single mental model of working with documents across applications, and simplifies learning new programs.

This is how the designers of Xerox Star described one of these universal commands:

MOVE is the most powerful command in the system. It is used during text editing to rearrange letters in a word, words in a sentence, sentences in a paragraph, and paragraphs in a document. It is used during graphics editing to move picture elements, such as lines and rectangles, around in an illustration. It is used during formula editing to move mathematical structures, such as summations and integrals, around in an equation.¹³

However, depending on the type of media application and the kinds of objects a user selects, “copy,” “cut” or “paste” will trigger different algorithms. For example, copying a phrase in a text document requires different sequences of operations than copying a selection in a bitmap image because the first is a one-dimensional sequence of characters, while the second is a set of pixels in a two-dimensional area. And even within a single application, many different algorithms will be needed to copy different kinds of objects a user can select.

My second example of how implementation of a media-independent technique involves different algorithms that work with particular media is the generation of random objects. The algorithm which generates a sequence of random numbers is very simple—it just calls the random number generator (a function available in all programming languages) to generate enough numbers, then scales these numbers within limits specified by the user (for instance, 0 to 1). This part is media-independent. Different applications can use this random number generation function as part of media-specific algorithms (i.e., algorithms which work on particular data structures) to create different types of content. For example, Photoshop has a command called “Add Noise” (located under Filters > Noise)

¹³ D. Smith, C. Irby, R. Kimball, B. Verplank, B., E. Harslem, “Designing the Star User Interface,” *Byte*, vol. 7, issue 4 (1982), p. 242–82.

A diagram of the Xerox Star UI from D. Smith, C. Irby, R. Kimball, B. Verplank, B., E. Harslem, “Designing the Star User Interface,” Byte, vol. 7, issue 4 (1982), 242–82. The universal commands are located in the dedicated keyword on the left part of the keyboard. (The original illustration from the article was redrawn in Illustrator.)

which generates a set of random X, Y number pairs. It then uses them to select specific pixels in the image and convert them to black or a random primary color (depending on the chosen option). A 3D modeling application can use the same technique to generate a set of identical 3D objects randomly located in space. Sound editing software can generate random sonic noise; and so on.

The implementation of media-independent techniques is structurally similar to various aesthetic systems in art that were not limited to a particular medium: for instance, baroque, neo-classicism, constructivism, post-modernism, and remix. Each system manifested itself across media. Thus, Baroque aesthetics can be found in architecture, sculpture, painting, and music; constructivism was applied to product design, graphic design, clothing, theatre, and possibly poetry and film.¹⁴ But just as with media-independent techniques, realizing a particular aesthetic system in different media required some specific artistic devices that explored possibilities and worked with the limitations of each medium.

File formats

Software uses files to store and transfer data. For example, when you save a Photoshop image, all its channels, layers, groups, paths, and other information written to file using a particular format. Along with data structures, algorithms and UI, *a file format is* another fundamental element of computational media. File formats are the standardized mechanisms for storing and accessing data organized in a particular structure. Some file formats like .rdf are in public domain; some like .doc are proprietary. As I will discuss in more detail in the “Design Workflow” section in Chapter 5, standardization of file formats is an essential condition for interoperability between applications that in turn affects the aesthetics of media created with these applications. From the point of view of media and aesthetic theory, file formats constitute the “materiality” of computational media—because bits organized in these formats is what gets written to a storage media when a file is saved, and also

¹⁴ Vlada Petric, *Constructivism in Film – A Cinematic Analysis: The Man with the Movie Camera* (Cambridge: Cambridge University Press, 1993).

because file formats are much more stable than other elements of computational media (I will explain this below).

Since both the materials and tools of physical media are now implemented as software, in theory new file formats and new algorithms can be easily created at any time, and existing ones can be extended. (Recall my discussion of “Permanent Extendibility” in Chapter 1.) However, in contrast to the 1960s and 1970s, when a few research groups were gradually inventing computational media, today software is a big global industry. This means that software innovation is driven by social and economic factors rather than by theoretical possibilities. As long as file formats are kept constant, it is easy to add new tools in subsequent software releases and the old tools will continue to work without modification. Moreover, in contrast to the period when Kay and others were defining “the first metamedium,” today millions of individual professional users—as well as design and architecture firms, film studios, stock agencies, web design companies, and other creative companies and groups around the world—store their work and their assets (3D models, photographs, print layouts, websites, etc.) as digital files in particular formats: .doc, .pdf, tiff, .html, etc. If file formats were to change all the time, the value of media assets created or owned by an individual or a company would be threatened.

As a result, in practice file formats change relatively infrequently. For example, the JPEG image format has been in use since 1992, while the TIFF format goes back to 1986. In contrast, the modification of software tools which can work on files—creating, editing, displaying, and transmitting them—and the creation of new tools happens at a fast pace. When a company releases a new version of its application software, it usually adds various new tools and rewrites some of the existing ones but the file format stays the same. This stability of media file formats also allows other developers (both companies and individuals) to create new tools that work on these formats. In other words, it is one of the conditions that make “constant extendibility” of media software possible in practice. Here is an example of this extendibility: when I visited the plugins area of Adobe’s Photoshop website on August 5, 2012 it listed 414 plugin products.¹⁵ Given that a typical product can include a dozen or even

¹⁵ <http://www.adobe.com/cfusion/marketplace/index.cfm?event=marketplace.categories&marketplaceId=28&offeringtypeid=5> (August 5, 2012).

thousands of filters, presents, or action sets, the total number of available plugins is likely to run into the hundreds of thousands.

Each file format and its corresponding data structure has its strengths and weaknesses. A photograph represented as a bitmapped image can be given a painterly appearance; blurred and sharpened; composited with another photograph, and so on. All these operations are much more difficult or even impossible with a vector image. Conversely, it is much easier to edit complex curves if they are internally represented by mathematical formulas—a format used by vector drawing programs such as Illustrator and Inscape. Because many projects call for the combination of effects only possible with different data structures (such as raster and vector), over time professional software applications were extended to handle the corresponding file formats in addition to their native format type. For example, while the majority of Photoshop CS4 tools are geared towards raster images, it also includes some tools for working with vector drawings. Photoshop can also import vector graphics, while Illustrator can import bitmap images.

However, this hybridization of software applications does not change the fact that each separate application tool can only work on a particular data structure. This is true for universal commands such as “cut,” “copy,” “paste,” and “view,” as well as for a multitude of media-specific commands such as “word count,” “blur,” “extrude” and “echo.” Thus, behind both media-independent and media-specific tools are separate algorithms each designed to work with particular data structures. However, a user has a different understanding of the two types, since the implementation is not visible directly. The former bring all media types together conceptually, and even creates an imaginary horizon where all differences between them disappear; at the same time, the latter emphasizes these differences since they only become available when a user works with a particular media.

As the name indicates, computer “files” refer to the paper files that were the key information management technology of a mid-twentieth-century office when computers were developed. The word “file” was used already in 1950 in RCA advertisement for its new “memory” vacuum tube; in 1952 the word was used to refer to information stored on punch cards.¹⁶ With the devel-

¹⁶ http://en.wikipedia.org/wiki/Computer_file

opment of the Web in the 1990s, web “documents”¹⁷ such as web pages became equally important. A web page may consist of a single HTML file that contains static text, and other media content stored on a server. Alternatively, a web page may be “dynamic,” which means that is constructed when the user accesses its address; it can also change as a result of user interaction.¹⁸ Dynamic web pages can be constructed using client-side scripting (e.g. JavaScript); they can be also constructed using server-side scripting (PHP, Perl, Java, and other languages). The two methods can also be combined using Ajax techniques; for example, the popular web application that uses Ajax is Google Earth. In 2011, HTML 5, the next generation of the HTML standard, enabled multimedia and graphics elements including video, audio, and SVG graphics without the need for client side plugins. As these and other technologies were gradually developed and adopted, the identity of the web has been gradually changing—from static pages in the first half of the 1990s to “rich internet applications” that match much of the functionality of traditional desktop applications.¹⁹

Given the complexity and variety of web documents and applications types, the multitude of technologies and techniques for creating them, and the continuous evolution of both technologies and web conventions, it would not be appropriate to simply mechanically map our concept of media data structures to the web. If we instead focus more on the meaning of data structure as a mental model of media shared by designers and users, as opposed to its technical implementations, then this concept does apply to the web documents and applications. However, rather than referring to the type of media and its characteristics (text, bitmap image, vector image, sound, 3D model, etc.) that become elements of a web document or an application, we can also use it to describe the *interaction possibilities and conventions* offered by a web document or an application. For example, web pages typically continue to have hyperlinks that allow a user to go to related pages. Today a web page may contain “social media buttons” which allows users to easily share some content on the page. Particular genres of web documents and applications offer

¹⁷ http://en.wikipedia.org/wiki/Web_document

¹⁸ http://en.wikipedia.org/wiki/Dynamic_web_page

¹⁹ http://en.wikipedia.org/wiki/Rich_Internet_application

their own interaction possibilities: for example, a blog typically contains a list of blog posts organized by dates; a webmail application contains buttons for responding, forwarding, and archiving email; and so on.

Although in principle I can go through all the most widely used types of web documents and applications, and write down a list of their conventions as they exist right now, this list will be both very long and no longer accurate by the time my book manuscript comes out in print. This is one of the reasons why this book focuses on discussion of media as *representation* rather than as *communication and interaction*—because the structures of software-based representations are more stable, less numerous, less complex, and change much less frequently than software and network based communication and interaction technologies. This does not mean that I give up on the project of *understanding web media* (including *mobile applications*, which currently number in the hundreds of thousands)—instead, I hope to do it more comprehensively in the future, while limiting myself to only brief discussions in this book.

Parameters

As I suggested earlier, a user's mental model of media creation and editing—both in the context of a particular application type, and when dealing with “digital media” in general—contains two fundamental elements, which, both conceptually and practically, correspond to two elements of computer programming: algorithms and data structures. When a user selects a particular tool from a menu and uses it on the part of the document s/he is working on, the algorithm behind this tool modifies the data structure that stores the content of the part. Most users, of course, do not know the details of how media software works on a programming level, and they may be only vaguely aware of what an algorithm is. (In 2011 I was driving towards San Francisco and saw a big ad board along the road, which prominently featured the word “algorithm”—but I could not stop to snap a picture that I really wanted to include in this book.) However, unknown to them, the principles of contemporary computer programming are “projected” to the UI level—shaping how users work with media via software

applications practically, and how they understand this process cognitively. The data structure/algorithm model is one example of this. We will now look at another example of such “projection”: options and their implementation.

One of the principles of modern computer programming—regardless of programming paradigm and programming language—is the use of parameters.²⁰ The popularity of parameters (which are also called “variables” or “arguments”) is due to several reasons. One is the common modern programming practice of breaking a program into separate functions. If a program has to execute the same sequence of steps more than once, a programmer encapsulates this sequence in a single function which can be then evoked within the program by its name as often as needed. (Depending on the programming language, functions maybe be called procedures, methods, subroutines, or routines.) Dividing a large program into separate modular functions makes it easier to write, read, and maintain. (This programming paradigm is called Procedural Programming.) Functions that perform conceptually related tasks (for instance, generating graphics on the display screen) are collected in *software libraries*; such libraries are available for all popular programming languages and their use greatly speeds up software development. A function definition typically includes a number of parameters that control the details of its execution. For example, a function that translates an image into another format will have a parameter specifying whether the output format should be JPEG, PNG, TIFF, or another format. (And if you choose JPEG, you will get another parameter to specify the level of compression). The second reason for the popularity of parameters is that many functions (and whole programs) solve mathematical equations. A formula defines a relationship between variables. For example, a sine formula looks like this: $y = A \cdot \sin(w \cdot x + O)$, where A stands for amplitude, w is frequency, and O is phase. If we implement this formula as a software function, this function will have parameters for each variable (i.e., w , x and O).

If you do not program, you may still be familiar with the concept of parameters if you use formulas in Excel or Google Docs spreadsheet. For example, to generate a column of random

²⁰ [http://en.wikipedia.org/wiki/Argument_\(computer_science\)](http://en.wikipedia.org/wiki/Argument_(computer_science))

numbers, you can fill its cells with the function `RAND()`. The formula does not have any parameters; it simply generates random numbers that fall between 0 and 1. If you want to generate random numbers which fall within a different range, use a different formula `RANDBETWEEN(bottom, top)`. This formula has two parameters which specify minimum and maximum values of the numbers of the range. For example, `RANDBETWEEN(10, 20)` generates a set of random values which lie between 10 and 20.

In a modern GUI the parameters, which control program execution, are called *options*. Users specify the values for the options using text boxes, sliders, buttons, drop-down lists, check boxes, and radio buttons. Sometimes the interface provides a few possible values and a user can only choose from them; in other cases she can enter her own value; in still other cases an application provides both possibilities. For example, a typical color picker allows a user to set R, G, and B values to create a unique color; alternatively s/he can choose from a set of predefined color swatches.

Use of options greatly expands the functionality of software applications by allowing the same application to perform a wider range of actions. For instance, imagine that you need to sort a set of numbers. Instead of using two separate programs to perform the sort in ascending and descending order, you can use a single program and set the parameter value, which would determine what kind of sort to perform. Or, imagine a round brush tool included in an image editing application's toolbox. You would not want to see a separate tool for every possible color, every possible brush radius, and every possible transparency setting. Instead, a single tool is given options which control color, radius, and transparency.

What does this mean for media theory? With softwarization, the implicit possibilities and different ways of using physical tools are made fully explicit. "Artistic techniques" and "means of expression" (see the definition of a medium which opens Part 1) are given explicit and detailed controls. Like all computer programs or functions, they now come with many parameters. For instance, Photoshop CS5's basic brush tool has these parameters: size, hardness, mode, airbrush capacity, opacity, and flow. Hardness, opacity, and flow can have any value between 0 and 100; there are 25 modes to choose from; and the diameter can vary from 1 pixel to 2500 pixels.

Do most users need all these options and such a degree of precision? Probably not. For example, controlling opacity in 5 percent intervals will probably be quite sufficient for most users. However, the algorithm that implements opacity behavior is exactly the same regardless of whether a particular parameter can have 10 settings, or 100. Since the general logic of software industry is to always try to offer users “more” than the previous application’s version or competitors, it is understandable that the brush interface gives us the larger number of options and a larger choice of their values, even though such precision may be not needed. The same goes for most other tools available in media software. In this way, the logic of programming is projected to the GUI level and becomes part of user’s cognitive model of working with media inside applications.

While adding more options does require additional programming labor, offering more values for these options typically does not. If I want the brush tool to have a transparency option, I need to write new code to simulate this behavior. However, giving a user a choice of 20 or 100 possible values for transparency does not cost me anything. Like any other media tool a brush tool is an algorithm that takes some inputs and generates outputs by applying a number of computations to these inputs. In the case of a brush tool, the inputs are the values of the options set by the users, and the colors of the pixels over which the brush moves; the outputs are the new pixel values. The algorithm does not care what the particular input values are, and the number of steps required to execute them also does not change because of particular values.

Pre-industrial physical media tools did not have explicit controls. There were no numerical parameters to set on a pen, brush, or chisel. If you wanted to change the diameter of a brush, you picked a different brush. The industrial era introduced new types of media tools that were mechanical or electronic machines: the electrical telegraph, photo camera, film camera, film projector, gramophone, telephone, television, and video camera. Like all industrial machines they now came with a few controls. Physically these controls appeared as knobs, levers, and dials. The next generation of media—*software applications*—give explicit controls to all the tools they include. The tools that did not have explicit controls before now acquired them; the tools that had a few were given many more. (While machines made from mechanical parts can

have only a limited number of possible settings, software parameters typically can have practically unlimited range of values.)

The Language of New Media introduced the idea of “transcoding”—the mapping of the conventions and principles of software engineering to cultural concepts and perceptions. The explicit *parameterization* of all media creating and editing techniques implemented in software is a perfect instance of transcoding logic. Thus, the fact that Photoshop’s brush tool comes with a number of options controls and that opacity and flow can take any value between 0 and 100 is only partially related to the meaning of this command—a simulation of different physical brushes. The real reason for this implementation of the command lies in its identity as a computer program. (As I already discussed, in modern programming, programs and their parts are given parameters, which in many cases can take any arbitrary numbers as inputs.)

Along with being a good example of how principles and conventions of software development in general are carried over into media applications and our media lives, parameterization also exemplifies another trend. Seemingly different media activities—editing photographs, creating 3D game characters, editing a video, or working on a website design or a mobile application—become similar in their logic and workflow: select a tool, choose its parameters, apply it; and repeat this sequence until the project is done.

Of course, we should not forget that the practices of computer programming are embedded within the economic and social structures of the software and consumer electronics industries. These structures impose their own set of constraints and prerogatives on the implementation of hardware and software controls, options, and preferences (all these terms are just different manifestations of software parameters). Looking at the history of media applications and media electronics devices, we can observe a number of trends. Firstly, the number of options in media software tools and devices marketed to professionals gradually increases. For example, a significant number of Photoshop’s tools and filters have more options and controls than in earlier versions of the application. Secondly, features originally made available in professional products later become available in consumer-level products. However, to preserve the products’ identities and justify price

differences between different products, a significant difference in feature sets is continuously maintained between the two types of products, with professional software and equipment having more tools and more parameters than their consumer equivalents. Thus Photoshop has lots of tools, Photoshop Elements offers fewer tools and iPhoto and Picasa have fewer still.

All this is obvious—however, there is also a third trend, more interesting for media theory. Following the paradigm already established by the end of the nineteenth century when the Kodak company started to market its cameras accompanied by a slogan “you push the button, we do the rest” (1892), contemporary software applications and media devices aimed at consumers significantly *automate media capture and editing* in comparison to their professional counterparts. For example, during the 2000s many consumer digital cameras only offered automatic exposure; to get full manual controls one had to go to the next price category of semi-professional cameras. In another example, towards the end of that decade, consumer cameras started to incorporate automatic face and smile detection—features that were not available on expensive professional cameras.

Like any other type of automation, automatic exposure requires more computing steps than the use of manual settings; the same goes for applying “auto contrast” or “auto tone” commands available in media software. Thus, if we equate the use of computers with automation, paradoxically it is the consumers who fully enjoy its benefits—in contrast to professionals who have to labor over all these manual settings of all these controls... but of course, this is what they paid for: to achieve effects and results that built-in automation cannot deliver. At the same time, by offering more high-level automation in consumer-priced products, the industry unintentionally undermines the professionals’ skills. For instance, today a number of web-based applications and application plug-ins can take a portrait photo and improve contrast, correct skin tone, remove skin imperfections and wrinkles—all in one step. The results can be surprisingly good—maybe not good enough for the cover of *Vogue* but quite sufficient for posting the improved photo to one’s profile on a social network site.

Interestingly, in the beginning of the 2010s this trend was partly reversed. Because of the much larger size of the consumer market and faster release cycles, hardware and software makers started to

offer some new features first in lower-level products and then only later in more expensive products. For example, Apple's Aperture 3 (2010) added options which had already been made available earlier in a consumer-level iPhoto application in its 2009 release—Faces (face recognition) and Places (a system for identifying geo locations of the photos and locating them on a map interface).²¹

The metamedium or the monomedium?

Given fundamental similarity in how “mediums” function as implemented in software, a logical question arises: do we need to talk about different mediums at all? In other words, is the computer metamedium a collection of simulated, new, and yet-to-be-invented media, or is it one “mono-medium”? Are we dealing with *the metamedium* or *the monomedium*?

We now understand that in software culture, what we identify by conceptual inertia as “properties” of different mediums are actually *the properties of media software*—their interfaces, the tools, and the techniques they make possible for accessing, navigating, creating, modifying, publishing, and sharing media documents. For example, the ability to automatically switch between different views of a document in Acrobat Reader or Microsoft Word is not a property of “electronic documents” in general, but as a result of software techniques whose heritage can be traced to Engelbart's “view control.” Similarly, the ability to change the number of segments that make up a vector curve is not a property of “vector images”—it is an option available in some (but not all) vector drawing software.

As we have learned, every media software also includes at least some tools that are not media-specific—i.e. they are not limited to working on particular data structures like raster images or vector drawings. Originally conceptualized at Xerox PARC as ways to enable users to transfer the cognitive habits learned in using one application to other applications, today a small number of Xerox's “universal commands” have become a larger number of

²¹ <http://photo.net/equipment/software/aperture-3/review/> (March 4, 2012).

“media-independent” media tools and interface techniques which are shaping how users understand all types of media content.

Despite these fundamental ways in which distinct mediums become aligned conceptually and practically, I do not want to abandon the concept of different mediums altogether. The substantial differences between the operations for media authoring and editing supported by different data structures (in the sense of this term used here) is one reason to keep this concept. And here are three additional reasons:

1. “Mediums” as they are implemented in software are part of distinct cultural histories that go back for hundreds and often thousands of years. Electronic text is part of the history of writing; digital moving images are part of the history of a moving image which includes shadow plays, phantasmagoria, nineteenth century optical toys, cinema, and animation; a digital photograph is part of almost 200 years of photography history. These histories influence how we understand and use these media today.

Put differently, we can say that any film today exists against the horizon of all films ever made and a subset of those films which a particular person making or watching a particular film has seen in their life; similarly, any digital image exists against the horizon of all images in a “museum without walls” (André Malraux) of human visual history. A medium, then, is not just a set of materials and tools (whether physical, mechanical, electronic, or implemented in software) and artistic techniques supported by these tools—it is also an imaginary *database of all expressive possibilities, compositions, emotional states and dynamics, representational and communication techniques, and “content” actualized in all the works created with a particular combination of certain materials and tools.*

Systematic digitization of cultural heritage is gradually turning this imaginary database into a real one. By 2012, Google has digitized 20 million books, while artstor.org offered over one million images of art and architecture digitized by 228 museums and private collections. However, this process did not start with digital computers. In the early twentieth century the development of public art museums, illustrated art magazines and books, lantern slide lectures, and the academic study of arts in the universities made large numbers of previously created artworks directly visible and accessible to the public (as opposed to small numbers only

accessible to art patrons). For instance, American museums started to develop lantern slide collections of their holdings after 1865; in 1905 the University of California at Berkeley offered the first architecture history course that used lantern slides.²²

The digitization of cultural collections since 1990 started to gradually bring together the materials dispersed among all these already available resources, making them searchable and accessible through single websites. For example, by early 2012 Europeana²³ provided information and links to 20 million digitized cultural objects including paintings, drawings, maps, books, newspapers, diaries, music and spoken word from cylinders, tapes, discs and radio broadcasts, and films, newsreels and TV broadcasts, contributed by 1500 European institutions.²⁴ In the UK, the BBC's Your Painting project was set up to offer free online access to digital images of all 212,000 paintings from all UK national collections; this goal was achieved by 2013.²⁵ In the US, the Library of Congress provides access to dozens of digital collections from a single portal. The collections include 4.7 million high-resolution newspaper pages (1860–) and over one million digital images, such as 171,000 scanned photo negatives from the Farm Security Administration/Office of War Information program (1935–45).²⁶

These institutional digitized collections are supplemented by user-uploaded born-digital and digitized cultural artifacts. YouTube and other video sharing sites contain a substantial sample of all cinema history in the form of short clips. Flickr has a large number of photos of artworks taken by museum visitors around the world. Portfolio sites for professional media creators such as Coroflot.com and Behance.com contain millions of portfolios in art direction, exhibition design, illustration, interaction design, motion graphics, and other fields. Manga scanlation websites contain millions of fan scanned and translated manga pages (as of March 4, 2012, mangapark.com hosted 5,730,252 pages for 9020 manga series²⁷); Scribd.com hosts tens of millions of text documents (you will

²² http://en.wikipedia.org/wiki/Slide_library

²³ <http://www.europeana.eu/>

²⁴ <http://pro.europeana.eu/web/guest/news/press-releases> (March 4, 2012).

²⁵ <http://www.bbc.co.uk/arts/yourpaintings/> (January 31, 2013).

²⁶ <http://www.loc.gov/library/libarch-digital.html> (March 4, 2012).

²⁷ <http://www.mangapark.com/> (March 4, 2012).

most likely find this book there as well); the deviantArt online community for user-generated art hosts over 100 million submissions from 190 countries (Spring 2012 data).²⁸

Although the distribution of what is available in all these online archives is highly uneven in relation to the types of media, historical periods, countries, and so on, it is safe to say that today the cultural classics—i.e. works of famous film directors, cinematographers, graphic designers, media designers, composers, painters, writers, and so on—are all available online either in complete form or in parts. As a result, the idea of a “medium” in a sense of *all the creative works and possibilities realized so far using a set of technologies* has become quite real. As opposed to holding the imaginary database of key works in a particular medium in your head (not the most reliable place for such large-scale storage), you can now quickly consult the web to locate and study any of them. Further, you are not limited to the holdings of a single museum or a library at a time; instead, you can use Europeana, Artstor, or other large-scale collection aggregators to browse their combined metacollections. For example, while the National Gallery in London has 2,300 paintings, Your Painting site from BBC offers images of 212,000 paintings from all UK national collections.

2. We can also use the term “medium” to refer to a *presentation/interaction platform*. If you think of iOS and Android platforms (each containing mobile devices, an operating system, and apps) as examples, you are right—but I also would like to use the word “platform” in a more general sense. Medium as a *platform* refers to a set of resources that allows *users* to access and manipulate content in particular ways. Understood in that way, a white cube in a modern art gallery is a medium; so are a modern movie theatre, a print magazine, network television, and a DVD. (Note that just as iOS and Android are ecosystems that combine many elements, many older media platform work similarly. Movie theatres, film production, distribution, and publicity form the cinema platform; so do television production, distribution, and TV sets.)

This meaning of “medium” is related to the twentieth-century concept of “media” in Communication Studies, which came from Shannon’s Information Theory: the storage and transmission

²⁸ <http://en.wikipedia.org/wiki/deviantArt>

channels and tools used to store and deliver information. However, if the latter concept includes both storage and transmission channels and tools, my “presentation platform” puts more focus on the reception technologies. These technologies may include special spaces, architecture (for example, large media surfaces which are increasingly becoming part of buildings’ interiors and exteriors), sensors, lights, and, of course, media viewing/editing/sharing devices and apps. Presentation platforms also “program” particular patterns of behavior: one walks through and touches architecture; stays silent in a movie theatre; interacts with family members while watching TV, moves one’s body in front of an interactive wall; etc.

During the nineteenth and twentieth centuries, presentation platforms were closely tied up with particular types of media content. Art museums displayed paintings and sculptures (and later performances and installations); newspapers published texts and images; TV presented television shows, news programs, and films. The gradual addition of media viewing and playing capabilities to computers (and later laptops, mobile phones, tablets, media players and other computer-based devices) loosened this connection. Distribution, storage, and presentation of different media types were no longer tied up to particular technologies and particular presentation platform. (This separation parallels the similar process of “softwarization” I have discussed in detail: the tools and techniques for media authoring and editing have become liberated from their reliance on particular physical and electronic technologies.)

Today, we can access most types of media on every computer platform. You can view images, videos, text documents and maps inside email, in a browser, on your notebook, a PC, laptop, tablet, mobile phone, internet-enabled TV, or an in-car or in-flight entertainment system. What differentiates these devices is neither the types of contents they can play, nor the basic interfaces they provide for viewing and interacting with content, but rather the relative ease with which one can navigate various media.

For instance, my 2011 Samsung LCD-TV came with a full web browser—however it was much better at playing cable TV and Netflix content than web browsing experience, which was quite hard with a TV remote control. In another example, the relatively small screen of a mobile phone in the early 2010s, its less powerful processor, and its smaller amount of RAM makes it a less-than-ideal

platform for editing feature films or doing CAD (Computer Aided Design). At the same time, the same small size gives a mobile phone platform many advantages. In many countries it is socially acceptable for a person to engage in chat or send and receive SMS on their phone during a social meal—but doing it on a laptop would be inappropriate. The size of a mobile phone also makes it a perfect device for location-based social networking (e.g., Foursquare) and other services: recommending social events in a city, following friends on a map, playing location-aware games, and so on.²⁹

Similar to mobile phones, many platforms take advantage of their social settings by adding unique features. For example, some airline in-flight entertainment electronics systems (e.g., Virgin America's and V Australia's *RED* Entertainment System³⁰) allow passengers to chat with other passengers or participate in multi-player games with them.

As a media presentation and interaction platform, each type of computer-based consumer device has its differences. Currently (early 2013), some mobile platforms such as iOS do not allow users to save documents directly to the device; instead, files can only be saved inside their respective apps.³¹ Some media devices such as e-book readers, video players, audio players, digital billboards, and game consoles often can only play a few (or even just one) type of media content. (While consumer electronics companies are engaged in the ongoing “convergence” war, gradually adding ability to play all media types to most devices, this trend does not affect every single device.)

Given the differences in their physical appearance (size, weight, form factor), physical interface (touchscreen, keyboard, remote control, voice input, motion sensing) and media playback/editing/sharing/networking capabilities, it is tempting to think of each type of device as a different “medium.” As presentation/interaction platforms these devices provide distinct user experiences and encourage distinct sets of media behaviors (sharing location, working, chatting, etc.). However, we should also remember that since they all use the same technologies—computers, software, and networks—they also share many fundamental features.

²⁹ http://en.wikipedia.org/wiki/Location-based_service

³⁰ http://en.wikipedia.org/wiki/In-flight_entertainment

³¹ http://en.wikipedia.org/wiki/Mobile_platform

Given this, we may want to think of all these presentation/interaction platforms using the idea of “family resemblance” articulated by many nineteenth-century thinkers and later by Wittgenstein—that things may be “connected by a series of overlapping similarities, where no one feature is common to all.”³² But this would not be accurate. Prototype theory, developed by the 1970s by psychologist Eleanor Rosch and other researchers, may fare better. Based on psychological experiments, Rosch showed that for a human mind, some members of many semantic categories are better representatives of these categories than other members (or example, a chair is more prototypical of the category “furniture” than a mirror.³³)

If we consider the best current implementation of Kay’s vision augmented with networking and sharing capabilities as the prototype (i.e. the most central member of the category) “of the computer metamedium” (for me, my current Apple laptop would qualify as such a prototype, but any full-featured laptop will also do), then all other computer devices can be situated at some distance from the prototype based on how well they instantiate this vision. Of course, it is also possible to argue that none of the current computers or computer devices realizes Kay’s vision sufficiently, because casual users cannot easily program them and invent new media. (Appropriately, Kay named his 1997 Turing Award lecture “The Computer Revolution Hasn’t Happened Yet.”³⁴) In this interpretation, Kay’s Dynabook is the imaginary ideal prototype, and each realized computer device is situated at some distance from it.

3. Another important meaning of the concept “medium” is related to human sensory systems, which acquire and process information in different ways. Each sensory system contains sensory receptors, neural pathways, and particular parts of the brain responsible for further processing. Traditional human cultures recognized five senses: sight, hearing, taste, smell, and touch. Additionally, humans can also sense temperature, pain, positions of body parts, balance, and acceleration.

³² http://en.wikipedia.org/wiki/Family_resemblance

³³ Eleanor Rosch, “Cognitive Representation of Semantic Categories.” *Journal of Experimental Psychology: General* 104, no.3, (September 1975): pp. 192–233.

³⁴ <http://blog.moryton.net/2007/12/computer-revolution-hasnt-happened-yet.html> (March 5, 2012).

Because the concept of senses has been important for ancient and modern Western philosophy, Buddhist thought, and other intellectual traditions, the discussions of senses in relation to art and aesthetics have also been very extensive, so to enter them seriously would require its own book. Instead, I will limit myself to a short discussion of a more recent research in cognitive psychology that can be used to support the idea of multiple mediums: how different types of information are represented and manipulated in the brain.

The important question debated in cognitive psychology for forty years is whether human cognition operates on more than one type of mental representation. One view is that the brain only uses a single propositional representation for different types of information. According to this view, what we experience as mental images are internally translated by the mind into language-like propositions.³⁵ The alternative view is that the brain represents and processes images using a separate representational system. After decades of psychological and neurological studies, the current consensus supports this second view.³⁶ Thus, it is believed the brain operates on and maintains mental images as mental image-like wholes, as opposed to translating them into propositions.

If we accept this view that language, and images/spatial forms require the use of different *mental processes and representations* for their processing—propositional (i.e., concept-based, or linguistic) for the former, and visual/spatial (or pictorial) for the latter, it helps us to understand why the human species needed both writing and visual/spatial media. Different mediums allow us to use these different mental processes. In other words, if media are “tools for thought” (to quote again the title of a 1984 book by Howard Rheingold about computers³⁷) through which we think and communicate the results of our thinking to others, it is logical that we would want to use the tools to let us think verbally, visually, and spatially.

³⁵ <http://plato.stanford.edu/entries/mental-representation/>

³⁶ Tim Rohrer, “The Body in Space: Dimensions of Embodiment,” in *Body, Language and Mind*, vol. 2, eds. J. Zlatev, T. Ziemke, R. Frank, R. Dirven (Berlin: Mouton de Gruyter, 2007).

³⁷ Howard Rheingold, *Tools for Thought: The History and Future of Mind-Expanding Technology*, a revised edition of the original book published in 1985 (The MIT Press, 2000).

An hypothesis about the existence of propositional and pictorial representations in the mind is one example of a number of theories which all share the basic belief that human thinking and understanding are not limited to the use of language. For example, in 1983 Howard Gardner proposed a theory of multiple intelligences, which included eight different categories: bodily-kinesthetic, verbal-linguistic, visual-spatial, musical, interpersonal, logical-mathematical, naturalistic, and intrapersonal. Recall also that Alan Kay based the design of the GUI on the work of the psychologist Jerome Bruner who postulated the existence of three modes of representation and cognition: enactive (action-based), iconic (image-based) and symbolic (language). And while in Xerox PARC's implementation the appeal to the first modality was limited to the selection of objects on the screen using a mouse, more recently they were joined by touch interfaces and gesture interfaces. Thus, the interfaces of modern computers and computer-based devices are themselves prime examples of a multiple media at work—adding over time more media as interaction mechanisms, rather than converging to a single one, like written language (original UNIX and other OS from 1960s–1970s) or speech (Hal from Kubrick's 2001).

The evolution of media species

As we see, while softwarization redefines what mediums are and how they interact, it does not erase the idea of multiple distinct mediums. In fact, in contrast to the idea of “convergence” popular in the 2000s in discussing the coming together of computers, television, and telephony, I would like you to think of the computational media using the concept of biological evolution (which implies increasing diversity over time). And here comes the ultimate difficulty with continuing to use the term “medium” as a useful descriptor for a set of cultural and artistic activities. The problem is not that multiple mediums converge into one “monomedium”—they do not. The problem is exactly the opposite: they multiply to such extent that the term loses its usefulness. Most large art museums and art schools usually have between four and six departments which supposedly correspond to different mediums

(for example, San Francisco's Museum of Modern Art divides its collection into painting and sculpture, photography, architecture and design, and media arts³⁸)—and this is OK. We can still use unique names for different mediums if we increase their number to a couple of dozens. But what to do if the number goes into thousands and tens of thousands?

And yet this is exactly the situation we are in today because of softwarization. Extrapolating from the dictionary definition of a medium that opens Part 1, we can say that different mediums have sufficiently dissimilar representational, expressive, interactive and/or communicative capacities. (How much of a difference is needed for us to declare that we have two mediums rather than one? This is another fundamental question that makes it challenging to use the term “medium” in software culture.) Consider all the factors involved in gradual and systematic expansion of these capacities: “permanent extendibility” of software; the development of new types of computer-based and network enabled media devices (game platforms, mobile phones, cameras, e-book readers, media players, GPS units, digital frames, etc.) and the processes of media hybridization manifested in software applications, technology prototypes, commercial and artistic projects... Do we get a new medium every time a new representational, expressive, interaction or communication functionality is added, or is a new combination of already existing functions created? For example, does the addition of voice interface to a mobile device create a new medium? What about an innovative combination of different visual techniques in a particular music video? Does this music video define a new medium? Or what about the versions of Google Earth that run on a computer and support different types of layers, and the iPhone version which in 2008 only supported Wikipedia and Panoramio layers? Are these two different mediums?

If we continue to hold to our extended medium definition (extrapolated from the dictionary definition), we would have to answer yes to all these questions. Clearly, we do not want to do this. Which means that the conceptual foundation of media discourse (and media studies)—the idea that we can name a relatively small

³⁸ <http://www.sfmoma.org/explore/collection> (March 6, 2012).

number of distinct mediums—does not hold anymore. We need something else instead.

To explain this in an alternative way, when we considered only one aspect of media ecosystem—media design—we were able to adopt the traditional understanding of “medium” (materials + tools) to describe the operations and interfaces of application software. We did this by proposing this new definition: *medium = algorithms + a data structure*. Following this perspective, we can refer to simple text, formatted text, vector graphics, bitmap images, polygonal 3D models, spline-based models, voxel models, wave audio files, MIDI files, etc. as separate mediums. But even with this approach it was already difficult to cover the design of interactive applications and websites. When we start considering the larger ecosystem of the proliferating devices, network services, interface technologies, media projects, and over one million apps for mobile platforms available to consumers, the concept can no longer be stretched to describe them in a meaningful way. In this section, I will explore one way to think beyond “medium.”

As I suggested above, the model of multiple species related via evolutionary development that we can borrow from biology offers a plausible alternative. This model is useful for thinking about both media authoring/editing applications and particular media projects/products (which, after all, are also software applications but more specialized—if media applications are content agnostic, projects typically offer particular content). The key advantages of a “species model” over a “mediums model” are their *large numbers* (Earth contains many million species—at least for now); their *genetic links* (which implies significant overlap in features between related species); and the concept of *evolution* (which implies constant development over time and gradually increasing diversity).

Each of these advantages is equally important. Instead of trying to divide the extremely diverse media products of software culture into a small number of categories (e.g. “mediums”), we can think of each distinct combination of a particular subset of all available techniques as a unique “media species.” Of course, a software application or a project/product is not limited to remixing already existing techniques; it can also introduce new one(s), which may then reappear in other applications/products. Such new introductions can perhaps be thought as new genes (keeping in mind all the limitations of our use of these biological metaphors). If

contemporary sciences such as evolutionary biology, genetics and neuroscience can describe and map (or at least, work towards describing and mapping) millions of distinct species, 3 billions of DNA pairs in the human genome, and 100 billion neurons in a cerebral cortex, why cannot media theory and software studies deal with the diversity and variability of software culture by providing richer classifications than the kinds we currently use? (For inspirational examples from live sciences, see the Human Connectome Project to create a comprehensive map of the adult brain,³⁹ and even more ambitious, The Blue Brain Project to create a completely realistic software simulation of the brain, accurate on a molecular level.⁴⁰)

Equally valuable is the notion that related species share many features. Eighteenth- to twentieth-century aesthetic tradition, from Gotthold Lessing to Clement Greenberg, repeatedly insisted on opposing a small number of mediums to each other, thus seeing them as distinct and non-overlapping categories. This trend intensified in European modernism of the 1910s–1920s, when artists tried to reduce every medium to its unique qualities. To do so, they gave up representation and concentrated on the material elements thought to be unique to each medium. Poets, such as Russian futurists, were experimenting with sounds; filmmakers proposed that the essence of cinema was movement and temporal rhythm (French film theory of the 1920s⁴¹) or montage (Kuleshov's group in Russia); and painters were exploring pure colors and geometric forms. For example, one of the pioneers of abstract art, painter Wassily Kandinsky, published two articles in 1919 with the titles which clearly signal his program to articulate an artistic language consisting only of basic geometric elements: "Small Articles About Big Questions. I. About Point," and "II. About Line."⁴²

Although artists in the second half of the twentieth century systematically revolted against this trend, instead embracing

³⁹ <http://www.humanconnectomeproject.org/about/> (March 7, 2012).

⁴⁰ <http://bluebrain.epfl.ch/> (March 7, 2012).

⁴¹ See Jean Epstein, "On Certain Characteristics of *Photogénie*," in *French Film Theory and Criticism*, ed. Richard Abel (Princeton: University of Princeton Press, 1988), 1: pp. 314–18; Germaine Dulac, "Aesthetics, Obstacles, Integral *Cinégraphie*," in *French Film Theory and Criticism*, 1: 389–97.

⁴² Kandinsky further developed the ideas in these articles in his 1926 book *Point and Line to Plane*.

“mixed media,” “assemblage,” “multimedia art,” and “installations,” this did not lead to a new, richer system for understanding art. Instead, only a few new categories were created (e.g., the ones I just listed), and all the extremely diverse new art objects created by these artists were placed in these categories.

Evolutionary biology instead gives us a model of the much larger space of objects, which overlap in their identities. This model fits much better with my theory of software culture as a large and continuously growing pool of techniques that can enter into numerous combinations, in the form of applications, and projects/products created with them, or through custom programming. Obviously, the majority of these “media species” share at least some techniques. For example, all applications with a GUI interface designed to run on a full-size computer screen will use similar interaction techniques. To take another set of media species I have already analyzed as examples of media hybridity, Fujihata’s *Alsace*, Sauter’s *Invisible Shape*, and Google Earth all share a technique of embedding videos inside a virtual navigable space. Both Fujihata and Sauter construct unique interfaces to video objects in their 3D worlds. In contrast, Google Earth “inhabits” the YouTube video interface when you embed video from YouTube in a placemark (although you have some control over the embedded player characteristics⁴³). This ability to embed the YouTube video player offers yet another example of a technique now used in numerous websites and blogs; so is the use of APIs offered by most major social media services and media sharing sites.

Finally, another attraction of the “species model” is the idea of evolution—not the actual mechanisms of biological evolution on Earth as theorized and argued over by scientists (because these mechanisms do not fit technological and cultural evolution), but rather the image of gradual but continuous temporal development and increased variability and *speciation* (emergence of new species) this idea implies—without implying progress. Without subscribing to the theory of memes, we can name a number of ways in which new techniques are transmitted in software culture: new projects and products that are seen by other designers and programmers; scientific papers in computer science, information science, HCI,

⁴³ YouTube Help, “Embedded player style guide,” <http://support.google.com/youtube/bin/answer.py?hl=en&answer=178264> (March 7, 2012).

media computing, visualization and other related fields; and also the code itself. This last mechanism is particularly important for us because, in contrast to previous media, it is unique to software culture (at the same time, it has a direct parallel with genetic code). As new techniques for media editing, analysis, interaction, transmission, retrieval, visualization, etc. become popular, they are coded in multiple programming languages and scripting environments (Java, C++, JavaScript, Python, Matlab, Processing, etc.) and become available as either commercial or, increasingly, open source software libraries ready for download online. If a programmer working on a new application, website, or any other media project wants to use any of these techniques, s/he can include the necessary functions in his/her own code.

Traditionally, new cultural techniques were transmitted via imitation and training. Professional artists or artisans saw something new and they imitated it; pupils, assistants, or students would learn techniques from their teachers; art students would spend years making copies of famous works. In either case, the techniques moved from mind to mind and from hand to hand. Modern media technologies add a new mechanism for cultural transmission—the interfaces and manuals of media devices, which carry with them the suggested, proper, ways of using these devices. All these mechanisms are at work today as well (of course, speeded up by the web). However, they are also joined by a new one—the *transmission of cultural techniques via algorithms and software libraries*.⁴⁴ This does not necessarily mean that such transmission does not introduce changes—programmers can always modify the existing code to the needs of their project. More important I think is the fundamental *modularity* of cultural objects created via this mechanism. On the level of techniques, a cultural object becomes an agglomeration of functions drawn from software libraries—the DNA it shares with many other objects that use the same techniques. (While commercial media products also widely use content elements such as photos purchased from stock libraries, this kind of modularity

⁴⁴ Jeremy Douglass suggested that we can study the propagation of techniques across software culture by tracking the use of particular software libraries and their functions across programs. Jeremy Douglass, presentation at SoftWhere 2008 workshop, University of California, San Diego, May 2008, <http://workshop.softwarestudies.com/>

is not openly acknowledged.) Because techniques are coded as software functions, this cultural modularity is closely linked to the principle of modularity in modern computer programming—the concept that a program should contain a number of self-contained parts. If a media project or an app introduces a new technique (or techniques) and they appear to be valuable, often the programmers make them available as stand-alone functions which then enter the overall pool of all techniques available within the computer metamedium—and thus easing the ways for others to adopt this technique in creating new projects.

I would like to end this chapter by quoting the historian Louis Menand who explains that while prior to Darwin scientists viewed species as ideal types, Darwin shifted the focus to variation—which for me is an overarching reason for thinking about media in the software age through the terms of evolutionary biology:

Once our attention is redirected to the individual, we need another way of making generalizations. We are no longer interested in the conformity of an individual to an ideal type; we are now interested in the relation of an individual to the other individuals with which it interacts... Relations will be more important than categories; functions, which are variable, will be more important than purposes; transitions will be more important than boundaries; sequences will be more important than hierarchies.⁴⁵

⁴⁵ Louis Menand. *The Metaphysical Club* (New York: Farrar, Straus, and Giroux, 2001), p. 123.