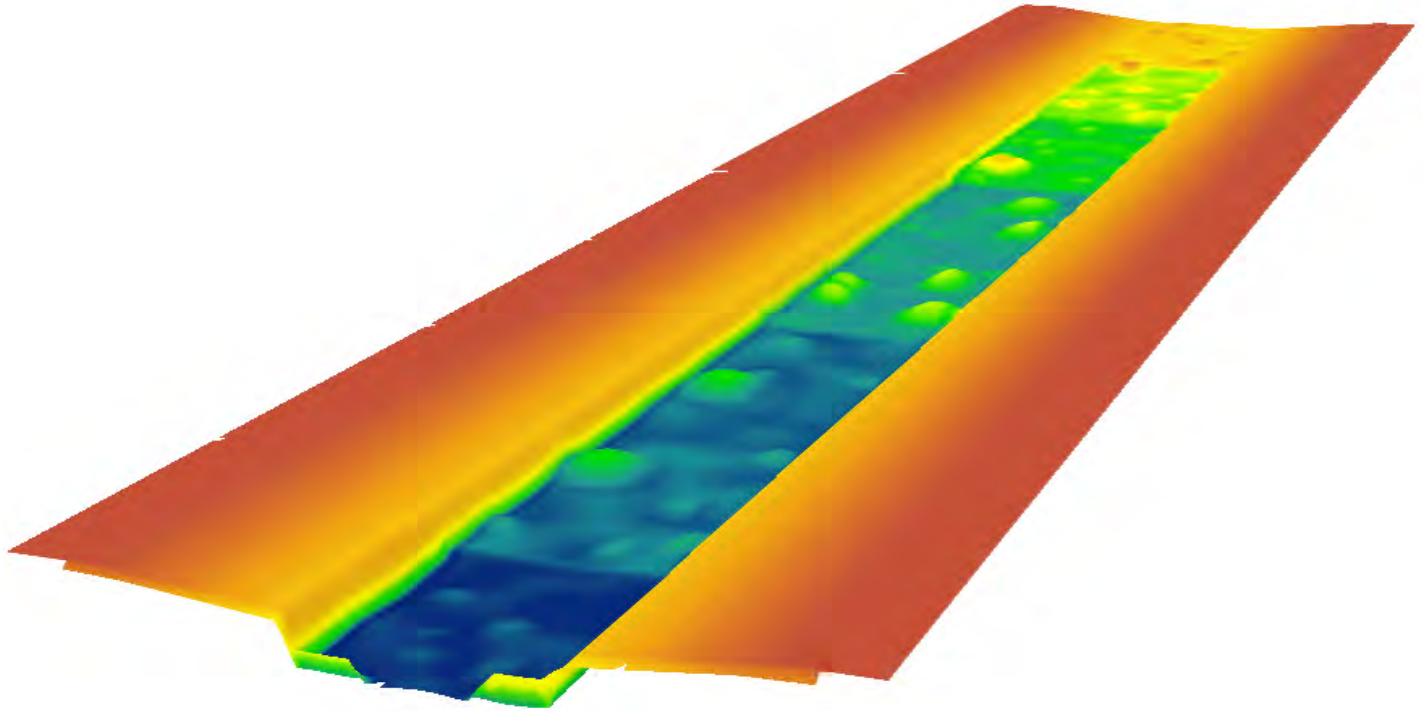


College of Agricultural & Environmental Sciences  
Department of Land, Air, and Water Resources  
UC Davis

# River Builder User's Manual For Version 1.2.0



May  
2021

By: Gregory B. Pasternack  
and Muwei Zhang

**UCDAVIS**

**Cite As:** Pasternack, G. B. and Zhang, M. 2021. River Builder User's Manual For Version 1.2.0. University of California, Davis, CA.

Location (URL):

Creators:	Prof. Gregory Brian Pasternack and Muwei Zhang
Title:	River Builder User's Manual For Version 1.2.0
Publisher:	
Publication year:	2021
Resource type:	Text/Text
Description [Other]:	Software manual for the use of River Builder to create synthetic river valleys in Python.
Subjects:	geomorphology, river topography, river design, river engineering
Rights:	The Regents of the University of California, Davis campus, 2014-19

**DISCLAIMER:** No warranty is expressed or implied regarding the usefulness or completeness of the information contained in this manual or the associated software. References to commercial products do not imply endorsement by the authors. The concepts, materials, and methods presented in this manual are for informational purposes only. The authors have made substantial effort to ensure accuracy, but there is uncertainty and the authors shall not be held liable for calculations and/or decisions made on the basis of application of this software and manual. The information is provided "as is" and anyone who chooses to use the information is responsible for his or her own choices as to what to do with the data.

For information contact [gpast@ucdavis.edu](mailto:gpast@ucdavis.edu)

# 1 USER'S MANUAL PURPOSE

The purpose of this manual is to provide you with an explanation of River Builder 1.2.0 software that has the capability to design rivers that meet regional, reach-scale geomorphic expectations, but go far beyond that to have multiple scales and layers of organized sub-reach variability. There is nothing like River Builder. Certainly, there are still some river archetypes the software cannot produce so easily- notably anastomosing, and braided rivers. However, going from version 1.0 to 1.1, River Builder added a new tool to greatly simplify procedural generation of step-pool and cascade channel types, so that is good progress. Many other new river object features were added in that update, such as boulder clusters, bed roughness, and man-made structure of various type. Before we could produce a manual for version 1.1, another issue came up that we felt we needed to change, so we took care of that and here we are at version 1.2.0. Specifically, going from 1.1 to 1.2 introduced an important back-end change that simplifies the process of designing a river by “reverse engineering” equations from a real reference site. We also added new accessory tools to help use the program. As always, we welcome collaboration for future developments that expand the software’s functionality more through time.

In most cases we expect people will use this software to design examples of real rivers to yield improved natural outcomes, but in fact the underlying geometric modeling is capable of allowing you to let your imagination run wild and design rivers for other purposes, such as testing dysfunctional river archetypes and creating unnatural, imaginary rivers for fictional purposes (e.g., video games, movies, animations, etc.). Whether you need a digital river design to fix a degraded real river or to save yourself millions of dollars in manual artistic effort for your next 4K resolution open-world fantasy game, this software can be very useful to meet your needs.

# 2 INTRODUCTION

Designing alluvial river channels that behave naturally is a central challenge facing river scientists and engineers as well as animators and video game developers in the 21st century (Brown and Pasternack, 2019). Though organized and responsive to driving forces, rivers exhibit complex patterns and processes from the scale of an individual grain of sediment to that of a large continent. Despite roughly a century of research it remains highly uncertain as to which patterns and processes are most important to design and build explicitly versus which ones should be allowed to emerge on their own after construction. Too little research has focused on variability in rivers. Nevertheless, our understanding of the fluvial patterns and processes as well as our ability to quantify them is increasing rapidly. We can now design much more dynamic rivers than ever before.

It is beyond the scope of this manual to explain the entire scientific foundation of this software. Brown and Pasternack (2019) provide a thorough literature review on the history and diversity of approaches to designing rivers, so readers new to this content are directed there for an in depth presentation. This introduction section offers a simplified, generalized overview of the context of river design necessary for this software. The “teaching” component of the website at <http://pasternack.ucdavis.edu> has a lot of free educational video podcasts about rivers and related topics as well as web pages that provide more explanation of underlying concepts. There are also a growing number of educational videos on Gregory Pasternack’s YouTube channel. In December 2020, new videos were added there about the use of geomorphic covariance structures, which is a key theory for designing realistic rivers. Much more is available on the internet and in numerous

textbooks. Most people will likely seek to just get started with the software and learn on an as-needed basis as they move along, so that is understandable.

The current version of River Builder and this manual are provided for free and open-source at this Github website:

<https://github.com/RiverBuilder/RiverBuilder>

Before getting into how this software works, it is important to clearly state that this software has one and only one use- *to create river corridor designs*. That's it. You know what you want a channel to be, and once you specify that then this program will procedurally render that for you. If you do not know what you want, but you want some channels (with or without valleys containing them), you can always take a look at the established set of sample template designs we provide on Github. The main output of River Builder is a comma-delimited (.csv) point coordinate file. You can import this into GIS, CAD, hydraulic modeling, or other software to achieve other functionality.

## 2.1 River Architect Software

Because users also need to analyze rivers as well as create them, we have created a sibling software platform called "River Architect" to evaluate a river with respect to ecohydraulic, geomorphic, and economic considerations (Schwindt et al., 2020). River Architect is programmed in Python3 and is now available open source and free to the public on Github at the link below.

[https://riverarchitect.github.io/main\\_page](https://riverarchitect.github.io/main_page)

[River Architect continues to undergo additional development and we welcome collaborators.](#)

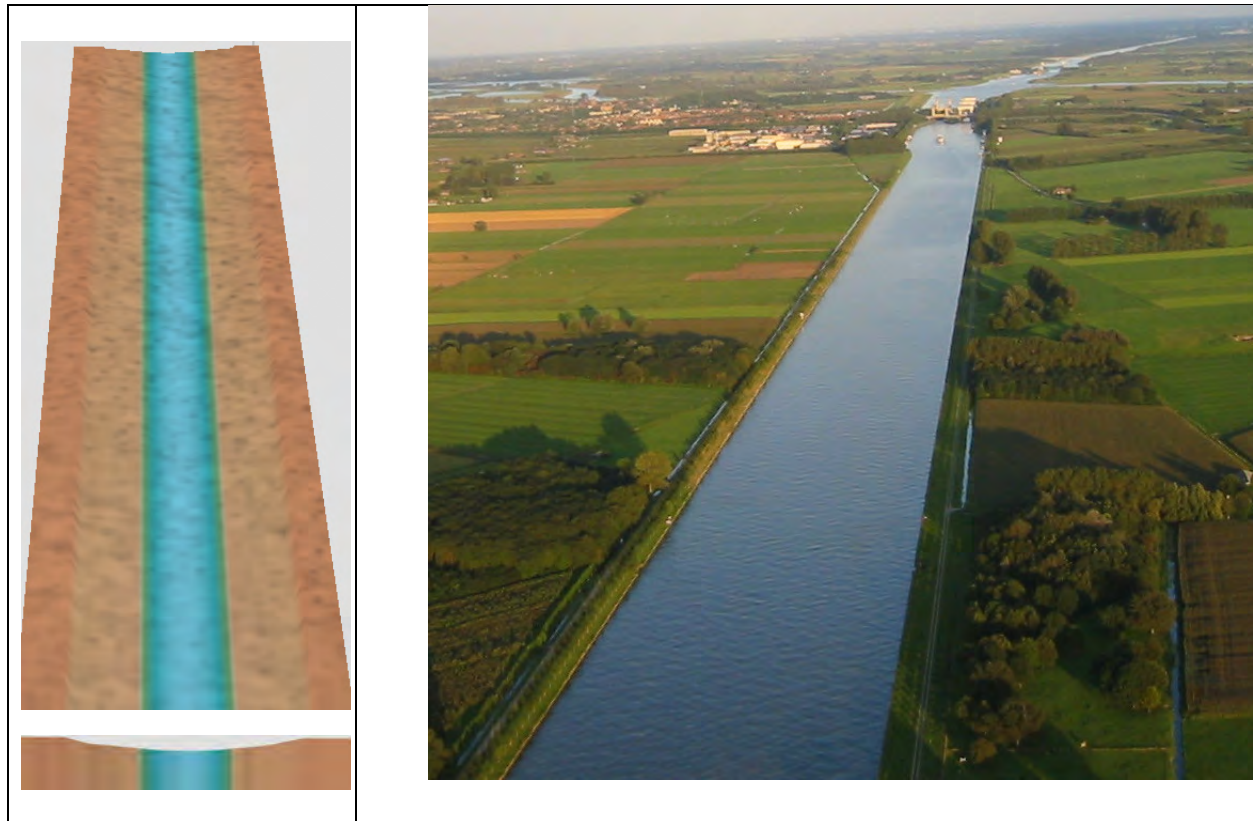
## 2.2 Vanilla Rivers

Decades of empirical study of longitudinal and lateral transects of alluvial rivers have yielded an explanation for the central tendency of channel geometric variables averaged over a length of 100-1000 channel widths (i.e., the reach scale) to grow or decline with discharge on the basis of mutual adjustment in order to pass the typical water and sediment delivered by the catchment. Such variables include slope, bankfull width, bankfull depth, sinuosity, entrenchment ratio, and median particle size. For any river reach there can be large uncertainties in the average values of geometric variables compared to empirical regional expectations. Typical uncertainties for channel width tend to be ~ 30-50% on average, but for individual locations can be much higher. Depth is more certain than width. How accurate geometric specification needs to be will vary depending on what geomorphic processes one seeks to instill, which is beyond the scope of this manual. Users are encouraged to use other software to test whether designed river terrain yield the processes and conditions in their design hypotheses (e.g., Pasternack et al., 2004; Brown et al., 2015).

A synthetic river designed only according to reach-scale metrics of central tendency is called a "*Vanilla River*" (Figure 1). Although vanilla can be a delicious flavor, it is colloquially considered plain, ordinary, without taste or distinction, and generally lacking in desirable variations and complexity. This is apropos, because in fact few river processes are driven by the central tendency of reach-scale river metrics. Instead, they are driven by local patterns of topographic variability. Thus,

many natural rivers do not look like synthetic Vanilla Rivers and we must turn to a more sophisticated understanding of topographic complexity to design rivers that reflect their natural patterning.

This software is capable of designing Vanilla Rivers, if that is desired. Vanilla Rivers, including ditches and canals, could be important to design for use in highly managed landscapes. For use in real rivers, we do not recommend stopping with only reach-averaged design metrics.



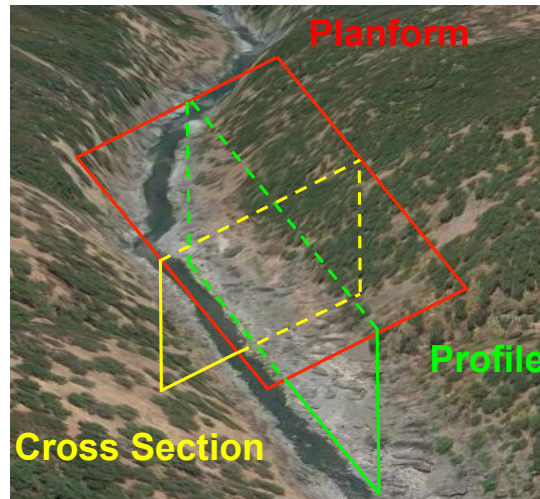
**Figure 1. Vanilla River examples, (a) synthetic, (b) real.**

## **2.3 A River's Three Planes**

Rivers are continuous three-dimensional (3D) features nested within 3D continuous mountain and valley landscapes. In the scientist's "reductionistic" approach to understanding, we often have to segregate nature into more discrete units we can understand and visualize in our mind's eye. River scientists do this by taking the three-dimensionality of a river and breaking it up into three independent two-dimensional (2D) planes (Figure 2). Following the concepts of Brown et al., (2014), River Builder uses this approach by specifying unique equations governing river topography in each plane. Then, the equations are used together to render a continuous 3D landform structure.

Throughout this manual, the terms plan view (aka planform or top view), profile, and cross section (aka XS) will be used to refer to these planes.





**Figure 2.** The major components of channel form represented as different planes after Knighton (1998) overlaid on an oblique section of the Eel River in California. Image from Google Earth. Copied from Brown et al. (2014).

## 2.4 Past Software Implementations

The underlying equations needed to make synthetic river valleys are already known, published, and non-proprietary. How those equations are organized into software is what is at issue here. Prior to the development of this version implemented in Python software, SRV algorithms were produced at UC Davis in three different software platforms for different purposes. First, there was the “RiverSynth” approach using Microsoft Excel<sup>®</sup>. This approach reproduces the examples in Brown et al. (2014) and has received some subsequent development with a few other features. It is very important to understand that these files can only create valley-orthogonal systems, and the equations are hard-wired with a valley-centric mindset. The formula in individual cells is unaware of and incapable of determining cross-sections that are perpendicular to the channel when the channel is not straight, so any lateral channel metrics for a sinuous centerline are not truly stream-wise.

Second, using an unrestricted donation, the Pasternack group at UC Davis sponsored World Machine, LLC to incorporate SRVs into the World Machine platform on the hopes that this would make this methodology more accessible. World Machine (<http://www.world-machine.com>) is a native geometric modeling platform for digital terrain development that allows for precise specification of parameters to design diverse landscapes through dialogue boxes and flowcharting. Implementations of World Machine with river design capabilities may not be available for free and may still be developmental, with little to no technical support. Be sure to touch base with World Machine, LLC to find out the current status of SRV tools in that platform as well as what support they offer to users to help learn their implementation of SRV tools and to address any bugs or problems that come up with your projects.

Third, the Pasternack group at UC Davis programmed a new implementation of the Brown et al (2014) equations in R and called it “River Builder”. This version has more features and capabilities than the Excel version, but is still using the same math, so it is subjected to the same constraints. This version is available as open-source and free to the public on the CRAN website at this link: <https://cran.r-project.org/web/packages/RiverBuilder/index.html>. It is also available with a

graphical user interface in River Architect software on Github.

Development of the Python3 version of River Builder began in summer 2019. Version 1.0 was released in May 2020. Versions 1.1 and 1.2 were finished in November and December 2020, respectively. Thereafter, it took a few months to update this manual, so this is now being formally released as of March 2021.

### 3 Important Concepts

#### 3.1 Channels Within Valleys


Channels exist within valleys and canyons. River Builder software can be used to make nothing more than a simple channel bounded by a wide, flat valley floor if that is all a user wants to have. However, the program was designed for creating entire synthetic river valleys to have maximum utility for a wide range of applications. This was chosen, because free-flowing rivers require lateral as well as longitudinal connectivity (Grill et al., 2019). Thus, a river designer should be mindful of lateral connectivity in their design, regardless of how far out from the centerline they intend to design. There are many examples of real-world scenarios in which a designer would want to overhaul an entire river valley (Figure 3). Such scope of construction is very challenging today due to the fuel cost for earth movement, but in a future society that maximizes its renewable energy potential, especially solar and wind, there would be no limits to energy usage and so massive earth movement for environmental restoration would become readily feasible, especially if performed by autonomous construction drones. Therefore, we need to provide the tools now for the scope of river restoration engineering that society will need when it awakens to address the global ecological collapse that is underway.

To prepare for use of River Builder, consider two ways channels are nested within valleys.

## Why Would You Wholesale Build a River Valley?

### Complete Procedural River Valley Design

- Valley filled with mine-waste
- Urban pipe daylighting
- Channel re-location
- Flood blow-out
- Mountain meadow pond-and-plug
- Research and Experimentation



Sierra Mountains, CA, USA

### **Figure 3. Examples of why a user might design a river valley, not just a channel.**

#### **3.1.1 Planform nesting**

One aspect in which channels need to be nested into a valley relates to their respective centerlines when view in overhead plan view. A simple scenario would involve a linear channel nested within the center of a linear valley (Figure 1a). However, there is no reason to force this. In the real world, channels meander and so do valleys (Figure 4). River Builder is capable of handling dual nested coordinate systems. The challenge lies in the order of operations to make this happen and get the outcome you want.

When you look at Figure 4, there are two ways of seeing/understanding the river's centerline. First, you can close your mind to what the valley is doing and assume that the channel is meandering in a universal, independent way regardless of the valley. From this viewpoint, valley and channel meandering are not mindful of each other and could be referenced to a single, universal coordinate system, which you might think of as a simple 2D (XY) horizontal Cartesian ground plane (aka planform plane, Figure 2). In that case, the user has to carefully insure that the two meandering functions do not conflict, but otherwise their math is referenced to the same coordinate system just with different functions.

The concern with this approach though is that as geoscientists we can think of ways in which the river's meandering IS influenced by the valley's meandering. If in fact there is a conditional dependence of the channel's finer, nested centerline on the valley's coarser outer valley bottom "belt", then the math of this approach would be wrong.

Second, you can view the channel as fundamentally nested within and dependent on the valley's structure. From this viewpoint, the valley and river each would have its own coordinate system, with the river's coordinate system nested within the valley's coordinate system. Mathematically, the procedure would involve first specifying the channel's centerline path and then nesting that within the valley's centerline path. The hard thing to understand and visualize about that, though, is that the effect of nesting one within the other means that from one point to the next along the channel, distances contort because the valley is bending. That means that the simple shape you envision for the channel deviates from that to take on what is required by the nesting. The effect is illustrated in Figure 5. What you can see is that as a designer, you envision a simple sine function for the channel's meandering, but then if this is nested in a curved valley, the channel's meandering warps to follow the valley's meandering. The more tortuous the valley meanders, the more warping the channel's meandering has to become to follow it. We can therefore show that the unusual plan view shapes meandering rivers take on can partly be explained (or at least mimicked) by the nesting phenomenon. In other words, you can design a channel's meandering using a sine function but when it is rendered it can look quite different from that.

The downside of this complexity is that it means that it is difficult to foresee how mathematical nesting of features will affect your final design topography. This is an endemic feature of River Builder for many of its capabilities; the software can produce highly complex, nested structures (i.e. landforms within landforms within landforms), so while you may start with a simple vision for how that would work, it can quickly get far beyond your expectations- at least until you get experienced with the software and come to know how different nesting structures pan out.



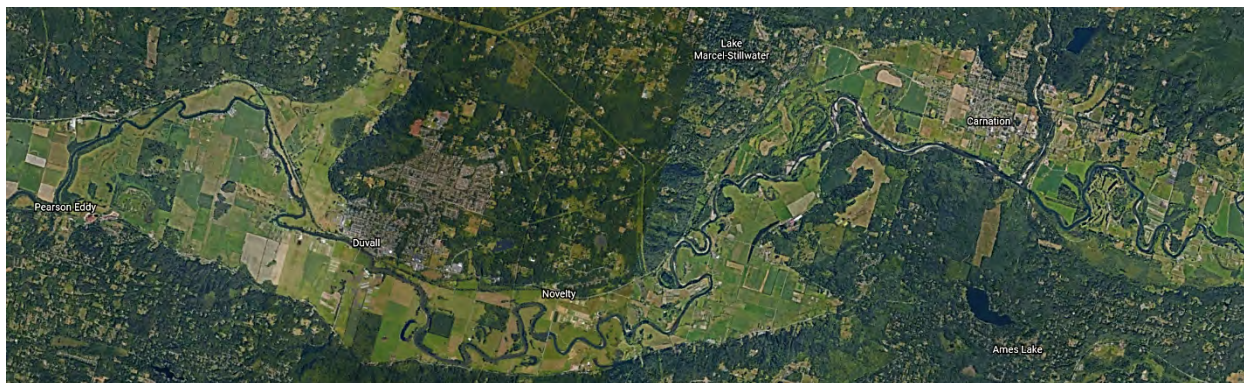


Figure 4. A real sinuous channel nested in a sinuous valley near Novelty, WA, USA. Each has a different sinuosity and the channel's lateral position is not locked to be at the valley's centerline.

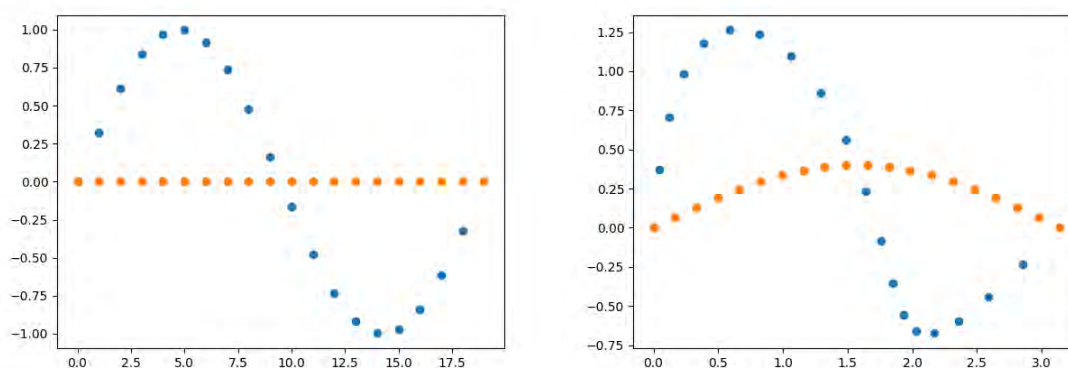


Figure 5. Same plan view as in Figure 4, but this time illustrating the steps and final effect of nesting the channel's centerline within the valley's centerline. The blue points are the path of a channel on the 2D horizontal (XY) ground plane (aka plan view), while the orange points are the path of the valley on that same plane. Left: you first create a channel's path assuming the valley is straight. Right: you then create the valley's path and nest the channel to move along it, but as a result of that the channel's path warps because equally spaced distances along the valley's centerline are no longer equally spaced distances on the 2D ground plane. The end product of River Builder would be the channel's path on the right.

### 3.1.2 Cross-sectional nesting

Sometimes a river corridor is nothing more than a single U-shaped conduit carved by water into a flat valley floor, such as shown in Figure 1a. However, valleys often contain far more cross-sectional features. Features typically have starting and ending positions across the valley's cross-section. These endpoints are commonly termed “**breakpoints**”. When these points are projected into the third dimension (up and down the river) along whatever path they follow, they are then called “**breaklines**”, even though they are not strictly lines.

In this manual, the term “**breakline**” will be used to refer to a plan view curvilinear function that denotes the starting and/or ending of a feature that runs down the river corridor.

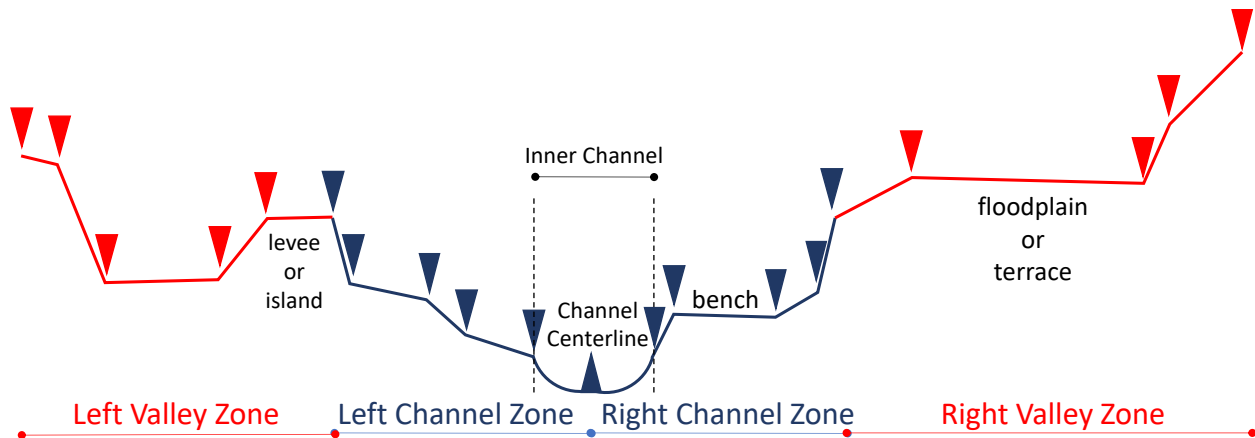
A simple conceptualization with a bit more complexity might be a U-shaped channel within a floodplain within terraces within steep valley walls or uplands (Figure 6). Within geomorphology, it has long been recognized that valleys may have multiple terraces as a result of distinct periods of higher or lower sediment supply and/or sediment transport capacity (Blum and Tornqvist, 2000). A terrace is a relatively flat surface positioned between floodplains and uplands. More recently, it has become increasingly recognized that channels too often have many distinct flat surfaces within them, which has led to the development of the term “macro channel” to refer to a ‘channel-in-channel’ form where “a smaller, low flow channel is inset within a larger channel, and separated from the margins of the macrochannel by geomorphic units such as benches, ledges, and various bar types” (Thompson et al., 2016).

In light of this natural range of possibilities, River Builder software aims to allow users unlimited freedom to create as many in-channel and in-valley flat surfaces and associated lateral slope breaks, aka breakpoints (Figure 7). Each lateral slope break is essentially an elevation contour (when projected into the longitudinal direction) herein termed a “breakline”. How this works will be explained later, but the basic concept is now established. It is even possible to create negative vertical offsets to obtain special features, such as levees, roads, and even multi-threaded channels (but such “yazoo” type channels may not intersect the main channel).

Because a design can have unlimited in-channel surfaces and lateral slope breaks, it is difficult to converse about some channel cross-sectional features in a universal way. Therefore, we coin the term “inner channel” to refer to the innermost conduit of water in the river valley (Figure 7). In some cases this may be the “bankfull channel”, but in other cases it may be a base flow channel or whatever else someone wants to put there with as many nested channel surfaces adjacent to it before reaching the floodplain’s overbank domain.



**Figure 6. River valley cross-section archetype. Copied from Eubanks (2004).**



**Figure 7. Conceptualization of the different cross-sectional zones (i.e. inner channel, left and right channel zones, and left and right valley zones) in River Builder. Within each zone, a user may specify an unlimited number of slope breaks (downward pointing triangles, aka “breakpoints”) to create whatever flat surfaces and side slope angles they want. This is done with user-specified horizontal and vertical offsets. Shown is just one example.**

Considering Figure 7 a decision has to be made about how to mathematically reference the position of each breakpoint relative to the others. There are three options possible. First, one can force the left and right side of the river to be symmetrical, in which case the term “width” can be used as a measurement and it would be centered on the channel centerline. Then, for each pair of breakpoints equidistant from the centerline on either side of it, the position of the point is simply half the width. The big downside to this option is that it forces both lateral and vertical symmetry. Width as a concept for design locks in a pair of breakpoints to share the same lateral and vertical distance from the centerline. As a software requirement, we wanted to allow users to be free of that constraint. Therefore, the term “width” becomes a fundamental limitation to imagination and river design that has to be abandoned when turning from traditional conceptualization and analysis of rivers to doing those in the modern era of high-resolution terrain capable of differentiating channel asymmetries. That’s going to be a new way of thinking for a lot of people, but remember- it frees you to explore natural asymmetry.

Second, one can allow breakpoints on the left and right side of the river to have independent positions AND the position will be specified as the position of the adjacent breakpoint closer to the centerline **plus** an incremental offset (Figure 8, left). For example if the channel bank top is at a position 3 m from the centerline and you want to specify the position of the outer edge of the floodplain before a terrace riser, then you would add the value 3 to the distance between the bank top and outer edge of the floodplain (for example, that is a distance of another 3 m). Consequently, relative to the centerline the position of the outer floodplain edge is 6 m. If the outer edge of the terrace is then 8 m further away, then the lateral offset would be 8 and the position of the outer edge of the terrace would be  $3+3+8=14$  (Figure 8, left). Thus, in this approach, one incrementally works outward and simply tracks the lateral offsets. Note that the same concept would apply in the vertical, too. This is the incremental offset approach.

River Builder versions 1.0 and 1.1 use this approach. There is nothing wrong with it. In fact, it has some very helpful mathematical advantages, which is why we used it. Specifically, because each position is offset from the next one closer to the centerline, then when we extrapolate a cross-

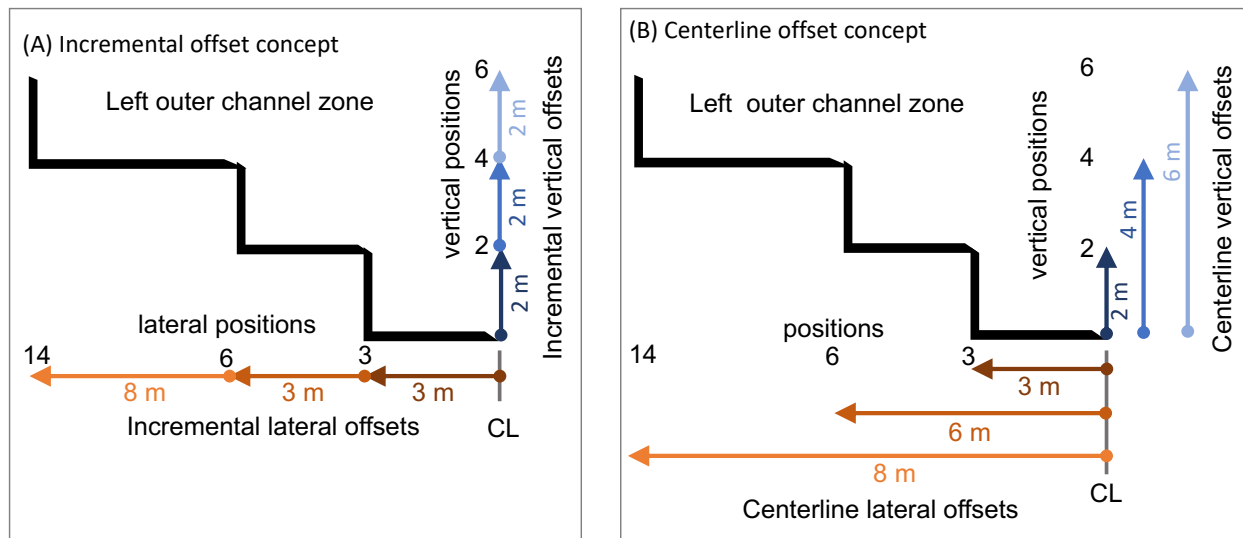
section's breakpoints into the profile plane to make a 3D representation, we can guarantee that contours following breaklines never cross over, no matter how complex a contour's shape is designed. That's very helpful and it saves users from having to check for cross overs manually. Note: if this is more complex than you are ready to think about right now, then don't worry about it. You can come back to this issue later.

However, as we progressed with software development we came up against a practical issue when we wanted to reverse engineer the nested, complex cross-sectional structure of real rivers to create archetypes of them for use in River Builder. You may know that in river design there is a common practice of identifying a "reference reach" to use as the basis for creating something new. There are pros and cons to this strategy, but that's not important right now. One use-case for River Builder is to generate essential river archetypes by reverse engineering equations from real rivers. The problem, however, is that it is very complex to reverse engineer breakpoint positions using incremental offsets. Basically, you have to figure out how to undo the nesting of quite complex functions. Instead, it is much better, faster, and easier to reverse engineer them by independently referencing each breakpoint to the centerline, both for lateral and vertical positions.

Thus, in the third concept, one can allow breakpoints on the left and right side of the river to have independent positions AND the position will be specified as distance from centerline (Figure 8, right). If you think back on the discussion about meandering, the approach there is an incremental, nested strategy; a coordinate system in a coordinate system. Consequently, it remains true that one cannot easily reverse engineer a nested channel centerline from a single coordinate system as it stands in River Builder 1.2. However, in that situation there are only 2 levels: channel and valley. As a result, it is not very hard to first reverse engineer the valley sinuosity, subtract that out of the spatial data series for the channel centerline, and then reverse engineer the channel's centerline. In contrast, a cross-section could have many levels of nesting making it very convoluted to try to do that on an incremental basis. It's possible, but wouldn't be pleasant, that's for sure.

Therefore, to aid reverse engineering, we have changed the offset referencing beginning with River Builder 1.2 so that each breakpoint's lateral and vertical positions are specified using the distance from the centerline. The only downside of this approach is that we have not written a routine in River Builder to check and insure that contours associated with different breaklines do not overlap; users will have to do that themselves. River Builder does provide nice graphs users can look at to quickly see if they have a problem or not, so they can iteratively fix any that arise. Thus, we have a trade-off right now, but it is worthwhile to make this change.

Because there is no other difference between versions 1.1 and 1.2, users currently have the option of which lateral and vertical referencing system they want to design with. If you prefer incremental offsets and an automated prevention of having your contours cross, then go with version 1.1. If you prefer to think in centerline offsets or you are reverse engineering a river from a reference using harmonic decomposition, wavelets, etc., then you'll want to use version 1.2.



**Figure 8. Views of the left half a river cross-section to help explain two concepts for specifying lateral and vertical offsets. (A) Incremental offsets used by River Builder 1.0 and 1.1; (B) Centerline offsets used by River Builder 1.2.**

Because the channel and the valley have independent centerlines, take note that the centerline lateral offsets are relative to one or the other centerline depending on what zone you are working on. Specifically, outer channel zone breaklines are offset laterally relative to the channel centerline. Valley zone breaklines are offset laterally relative to the valley centerline.

Having said all that, there is of course an exception! Ha! The one exception is the outermost valley boundary for your entire terrain. River Builder provides users with independent left and right outer boundaries; there is no terrain beyond these breakpoints/contours. We do want to guarantee that these do not cross over with contours inside the valley. As a result, the outer boundary breakpoints are set as an incremental offset from the next breakpoint closer to the channel centerline.

### 3.2 Steps & Waterfalls

River Builder 1.0 has a wide range of capabilities, but the problem for the average user is that it is quite technical to design some features working natively from mathematical equations. You might think we could keep adding more code to the software as needed for more features, but the problem is that it could become very convoluted to have different code bases specialized for different channel types. Instead, we think it is best to have a core mathematical framework of equations that we do not tinker with, and then we build stand-alone, complementary tools that you can use to design special river types or features. The way this works is that for each unique channel situation, we can develop a lexicon and set of inputs that are sensible to help users conceptualize and design that type of river. Then they run the stand-alone tool with that input to produce a text file. This text file output provides you with the text lines you need to copy/paste into the input file to run River Builder 1.2. It sounds a bit clunky, but it is actually worse to have an ever-increasing decision tree of input for the main program. This way keeps the main program the same and diverts new coding and features to the stand-alone tools.

The first major stand-alone tool we have produced is used to create stepped inner channels,



where there is any number of steps based on user specification (Figure 9). This tool is called “FunctionGenerator.py”. No coding skill is required to use it. The details of how to implement it are covered later in this manual. The goal here is to explain it conceptually.

In essence, the way a step is designed is by independently specifying its “tread” and “riser”. For example, if you want a step-pool design, then you can assign a pool-type longitudinal profile function to the tread. If you want to switch from a vertical step to a bed-supported slide, then you can specify a function for the riser with the linear slope you want. The specifications can be done for groups of steps, or if you want total control on a step-by-step basis, then you can have that, too. That’s all there is to you. Consequently, the tool is very easy to understand and use, if you know what you want for your design.

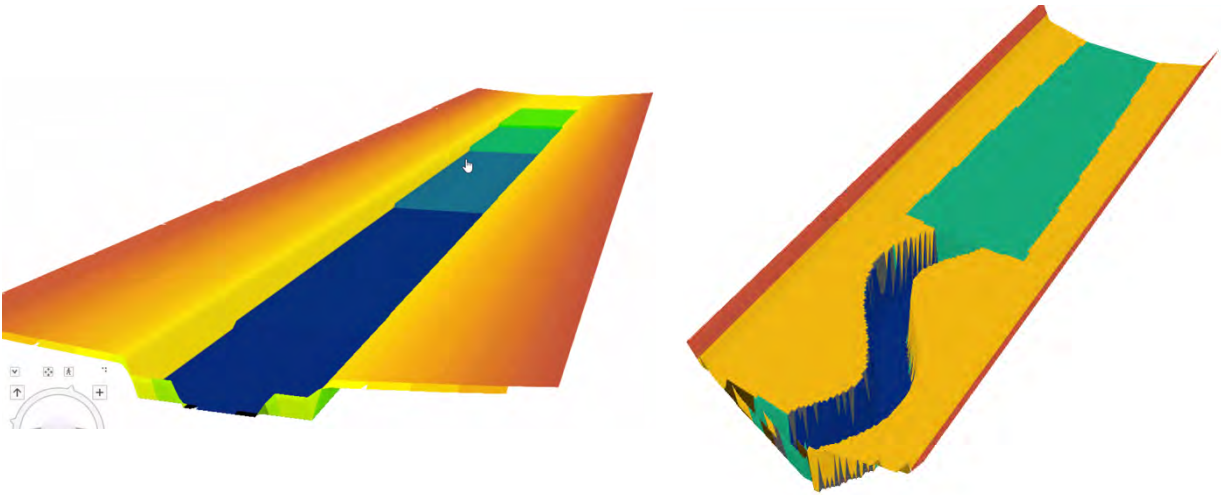
Making a stepped-channel inevitably puts some constraints on other possible features to add, but this is also realistic for natural rivers. First, steps are only used for the inner channel and the series of topographic points you get to render the steps are only along the inner channel centerline. As a result, it is highly recommended to choose a rectangular cross-section shape for the inner channel. That also means that one cannot nest channel steps within floodplain steps within terrace steps or anything like that.

Second, we recommend being careful when using stepped inner channels together with channel and valley sinuosity. Many stepped channel types are pretty straight, so keep that in mind. If you need sinuosity, it might be better to apply River Builder’s piecewise and masking functionality to achieve it rather than making a single reach with one function using `functiongenerator.py`.

Finally, there is an important bank issue when making steps. If you examine the left example in Figure 9, you will notice that the steps are increasingly entrenched as it goes from upstream to downstream. This happens because the river valley has a slope to it, while the steps may be quite a bit steeper than that slope, necessitating entrenchment. If the valley slope is high enough to keep up with the step drops, then that will not happen, but often that will not be the case.

Here is the concept of how this is computed mathematically. First, the code identifies the highest elevation of the channel centerline, which is likely the upstream-most centerline point. Next, the user specifies a reach-average channel depth in River Builder, so the program adds that value to the highest bed elevation to obtain the position of the inner channel bank top. From this point, the bank then is assigned a linear function with the computed river slope. Thus, if the steps are much steeper than the river slope, entrenchment must result.

The workaround - if you want to make a very long step reach – is simply to make individual short reaches, run them in River Builder, and then put them together in your GIS or CAD software later. That way, each new reach can reset the bank top coordinate to where you want it, and this will prevent deep entrenchment.



**Figure 9. Step topography examples. Step-run channel (left) and individual waterfall with a wide, shallow channel leading into a confined canyon (right).**

### 3.3 Geomorphic Covariance Structures

Many measurable variables in geomorphology and allied sciences vary along a pathway, such as a river corridor (Figure 10, top). Variables could be flow-independent measures of topography, sediment attributes, flow-dependent hydraulics, topographic change, and biotic variables. Lane et al. (2017) reported that for a large region of California river variability metrics distinguished channel types better than traditional central tendency river attributes. These variations can contain some random aspects, but to a large degree they are highly organized, interlocked, and readable (Brown and Pasternack, 2014, 2017; Pasternack et al., 2018a,b). Also, river variations can be layered on top of each other, yielding multiple spatial scales of topographic complexity. Both geomorphic processes and ecological functions are more strongly governed by layered scales of spatial variability in topography than the central tendency of a river averaged over scales. In turn, both geomorphic and ecological processes are vital to maintaining spatial diversity across scales.

Brown and Pasternack (2014) coined the term "Geomorphic Covariance Structure" (GCS) to mean the linked bivariate pattern of any two river variables along a pathway. It is not the statistical covariance, which is a single number, but instead a new concept involving the complete bivariate spatial series from which a statistical covariance could be computed if desired (Figure 10, bottom). For example, one GCS could be the spatial series of the product of depth and width, making it a surrogate for cross-sectional area. Note that detrended, standardized bed elevation ( $Z_s$ ) is a surrogate for depth. The GCS between  $Z_s$  and standardized width ( $W_s$ ) is the basis for the hydro-morphodynamic mechanism of flow convergence routing (MacWilliams et al., 2006; Pasternack et al., 2018a,b). Another GCS could be the spatial series of the product of channel centerline curvature and width. There are many potential GCSs one can envision. The theory of Geomorphic Covariance Structures (GCSs) is not only useful for assessing the layers of topographic patterning of real rivers (Brown and Pasternack, 2014, 2017) but also for the design of synthetic rivers with more natural landforms that drive the real diversity of physical processes (Brown et al., 2014, 2015).

A series of YouTube videos has been produced to help you understand GCS:

Part 1: <https://www.youtube.com/watch?v=VSMK72FbTfI>

Part 2: <https://www.youtube.com/watch?v=mZT3wbRAZZ4>

Part 3: <https://www.youtube.com/watch?v=tr82mvR-5kY>

Part 4: <https://www.youtube.com/watch?v=yssqRndHleQ>

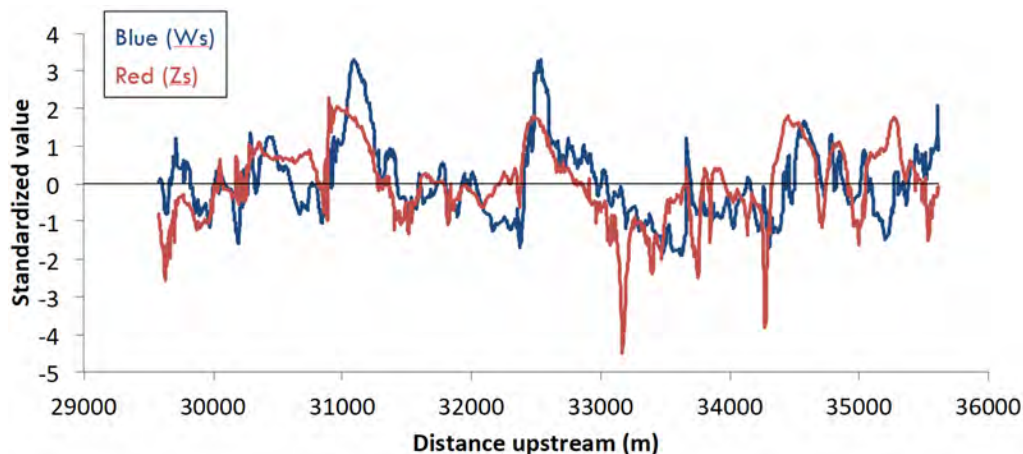
ISE2020 presentation: <https://www.youtube.com/watch?v=laoh6FysoNU>

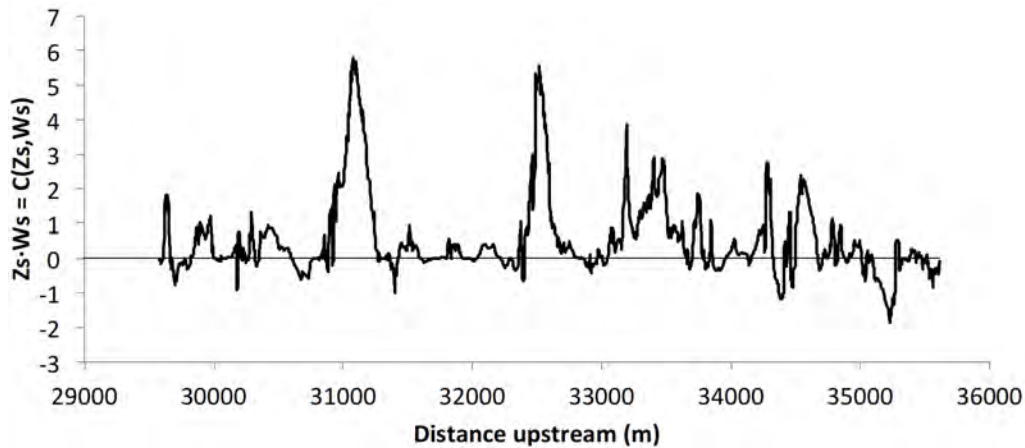
AGU2020 presentation: <https://www.youtube.com/watch?v=yvikf8C8Fxs>

This software is capable of implementing GCSs to produce rivers with organized, coherent patterns of variability. This is where the real power of this software lies. However, the software does not tell you what GCSs you need to produce different outcomes, nor does it tell you how to specify individual variability functions for each breakline to have it work in coherence with others to yield a specific GCS. You must have in mind what you want and an understanding of what GCS metrics are required to achieve that vision. Figuring that out is one of the primary open research lines in all of fluvial geomorphology today. It is the most important for advancing process-based river restoration. Take note that if you are highly uncertain if your design specifications will yield the geomorphic processes and ecological functions to seek, then it is important to undertake testing to evaluate design performance relative to your design hypotheses. An increasing number of predictive, mechanistic computer programs test river terrains this way, though they were developed to test real rivers, but nothing precludes their use for testing artificial river designs as well.

Over time we will aim to provide an increasing number of template files with different river archetypes you can use as a starting point, but that is still under development. As archetypes become available, they will be posted at the River Builder Github and within the web site at the link below:

<https://riverbuilder.ucdavis.edu/>





**Figure 10. Sample spatial series of detrended, standardized bed elevation ( $Z_s$ ) and standardized width ( $W_s$ ) (top) and an example geomorphic covariance structure (bottom), in this case the product  $W_s \cdot Z_s$ . Data taken from Pasternack et al. (2018b).**

### 3.4 Special Objects

Imagine you want a river with a single large boulder in it. River Builder 1.0 was not designed to cope with that. You could add such a “hero” feature to River Builder output in CAD or GIS, so we left that alone. However, there are significant benefits in allowing suites of natural objects to be designed in River Builder, including boulder fields, scour pools, and general bed roughness. Keep in mind that no features are possible at a finer resolution than the output points, so this is not a way to create sand-bed river texture or anything like that. We also decided to add some human-built structures as object feature options.

All of these types of discrete objects cannot be procedurally generated using the core code in River Builder, because that code is conceptualized around continuous functions, not objects. Therefore, River Builder 1.2 has a new set of core code that adds these capabilities with three types of objects: bed elements, bed roughness, and dams. This manual’s cover page image shows an example of a stepped channel with bed roughness and a few sizes of bed elements.

Note that all of these special objects only generate within the inner channel in River builder v. 1.2. They are added after the inner channel topography is calculated. This creates some constraints if you want to have a nested baseflow channel within a bankfull channel using objects across both. However, one can also apply an inner channel cross-sectional shape as a workaround for now.

#### 3.4.1 Bed Elements

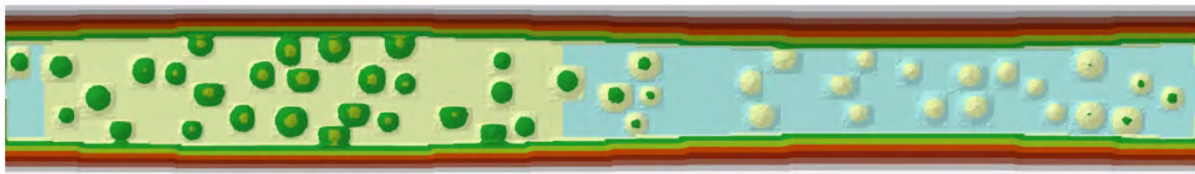
A “bed element” is defined herein as a roughly rounded square object, like a boulder or scour pool, so it can be protruding above the riverbed or making a hole down below it. Bed elements are organized into “bed element groups” based on a size class. A user may specify an unlimited number of bed element groups, each with its own characteristic size.

For a single bed element group, the user specifies the number of objects, their mean horizontal size, the standard deviation of horizontal size, and the characteristic height (Figure 11). Bed elements have a minimum horizontal size of 5, and a maximum size of  $\text{size\_mean} + 3 \cdot \text{size\_std}$ . The actual horizontal size of each bed element is computed at random using the normal distribution. This is done to create some natural variability within each group.

Meanwhile, height works a bit differently. A user specifies a height value that they want for each group, but then to create some natural variability, a small deviation from that is added to each individual. Specifically, random noise is added that is  $\leq \pm 10\%$  of the height value. For example, if a height value of 10 is specified, then each individual feature could be as short as 9 or as tall as 11. Height may be specified as positive to make a boulder/protrusion or negative to make a hole in the bed. A bed element may be higher than the inner channel bank depending on its height.

With those specifications, River Builder then randomly places the specified number of individual bed elements on the bed within the inner channel. The code will always try to find a place to fit the specified size until the designated number is reached or there is no more open space left in the inner channel. If that happens, it stops; bed elements within the same group *will not* overlap with each other.

While it is impossible to form a clustered bed element using a single group, this can be achieved with multiple groups, because bed elements in different groups *may* overlap with each other.



**Figure 11. Example of a simple inner channel with a single bed element group of boulders. Note that the base of each element is relatively square while the top is rounded.**

#### 3.4.2 Bed Roughness

Natural riverbeds are rough, by which is meant that the surface is textured and the topography is not a perfectly uniform elevation. River Builder 1.2 allows users to add bed roughness as a comprehensive coverage to the inner channel. Bed Roughness is a “random” Perlin noise added to the elevation of the inner channel bed. To simplify the specification of the function, a user simply provides a height range from a negative height to a positive height around the bed elevation. This enables vertical asymmetry such that the roughness protruding above the bed can be higher than the depression below the bed, or vice versa. For example, if you wanted 0.5 m of roughness but only 0.2 m of it protruding above the bed, then you would choose  $[-0.3, 0.2]$  as the inputs.



**Figure 12. Example of inner channel bed roughness.**

#### 3.4.3 Dams

In River Builder 1.2, a “dam” is nothing more than a nearly vertical wall blocking the inner channel bed. In river management, there are a wide variety of such simple barriers, such as sills,



weirs, check dams, etc. River Builder is certainly not a tool for designing sophisticated major dams, but this new feature adds more capabilities to the tool and was easy to add.

A user simply specifies the centerline X-position of the dam, the height, and the “thickness” of the dam. X-position is not the valley’s X position, but the real x coordinate after all centerline meandering is accounted for. Depending on how complex your centerline variability functions are, it may take some trial and error to get dams where you want them.

Dam height is how high above the bed elevation of the centerline you want the dam to be. It is a bit more complicated than that, necessitating more explanation. It is a bit more complicated than that, necessitating more explanation. Other than for a rectangular channel, a river’s cross-section has different bed elevations across the channel. Also, bed elevation at any location may be affected by the presence of a bed element or bed roughness. Usually, a dam has a flat crest at a specific elevation, so in order to achieve that River Builder over-rides all other elevation information where a dam is placed and the bed elevation becomes the value of the centerline bed elevation plus the user specified height. For example, if the centerline bed elevation is 3 m and you specify a dam 5 m high, then the elevation of the dam crest will be  $3+5 = 8$  m. If there is a 2 m boulder on the right side of the channel at this cross-section, then its height will be over-ridden with a bed elevation of 8 m to keep the dam crest horizontal. However, if the boulder protrudes upstream or downstream beyond the thickness of the dam, then the boulder’s elevation will appear in the adjacent points.

Finally, dam “thickness” refers to the upstream to downstream extent of the height. In other words, a high value yields a broad-crested dam and a low value yields a sharp-crested dam. The dam will be centered at the specified X-coordinate and extend upstream and downstream equally relative to that point. For a location at which the river’s centerline is straight, then the dam thickness will exactly equal the specified value. For a meandering river, the value a user specifies will get warped by the meandering, but the thickness will be the same across the channel. I may take some iterative trial-and-error to end up with the desired exact thickness.

A user may implement as many dams as desired, specifying the three parameters for each dam as a list.

## 4 Major Steps in Designing a Digital River

Brown et al. (2014) presented a seven-step method of river valley design (i.e. synthetic river valley (SRV) design) involving geometric modeling in which multiple scales of continuous equations are specified in each plane (XY, XZ, and YZ; Figure 2) and combined in a digital elevation model (DEM). From that starting point, the framework has further progressed over the years, generally adding more capabilities, yielding the current vision shown in Figure 13. To follow the schematic, start in the upper left and work counter-clockwise.

### 4.1 River Valley Conceptualization

First, you conceptualize the river corridor in terms of its essential elements and scales on the basis of eco-geomorphic goals. This is where you do your homework to figure out what it is you want to design and what your design aims to achieve. If your design is artistic, then you have vast freedom with the software to pursue your artistic vision. If your design is intended for use in real rivers or for scientific inquiry, then you should establish clear design hypotheses (as defined by Wheaton et al., 2004) and have the capability to translate those design hypotheses into flow-form-

function connections (e.g. Lane et al., 2018).

River Builder assumes a grid-type projected coordinate system with orthogonal {X, Y, Z} coordinates as the desired input and output. However, internally it transforms data into nested stream-wise and valley-wise coordinate systems so all computations go along the river and valley. In terms of units, it just takes numbers in and produces numbers out without any reference to units. The only exception is an optional routine to estimate bankfull channel depth from bed material grain size assuming that at bankfull discharge the bed material would be right at a state of incipient motion. Therefore, the outputs are infinitely scalable simply by specifying units of whatever scale you want. A width of 60 could be 60 inches, 60 mm, 60 football fields, 60 km, etc. It's all arbitrary to meet your needs. Therefore, when you import River Builder outputs into another software program, make sure you select an approach grid projected coordinate system with the units you want the numbers to refer to.

As part of the conceptualization process, you think through how you want to specify the geometric elements for each plane at all scales of interest that are going to be needed. All of these items are covered in detail later in the manual, but just for example purposes, they can include (but not limited to) channel centerline planform shape, channel bed elevation longitudinal profile along the centerline, XS channel shape, XS valley shape beyond the channel, channel topographic roughness and the planform profile of every key contour in the river valley you intend to instill. The planform shape of every key contour can be held constant or vary downstream according a function or additive set of functions. You can also conceptualize to add discrete objects to the inner channel, such as bed elements of various size classes and a channel-spanning dam.

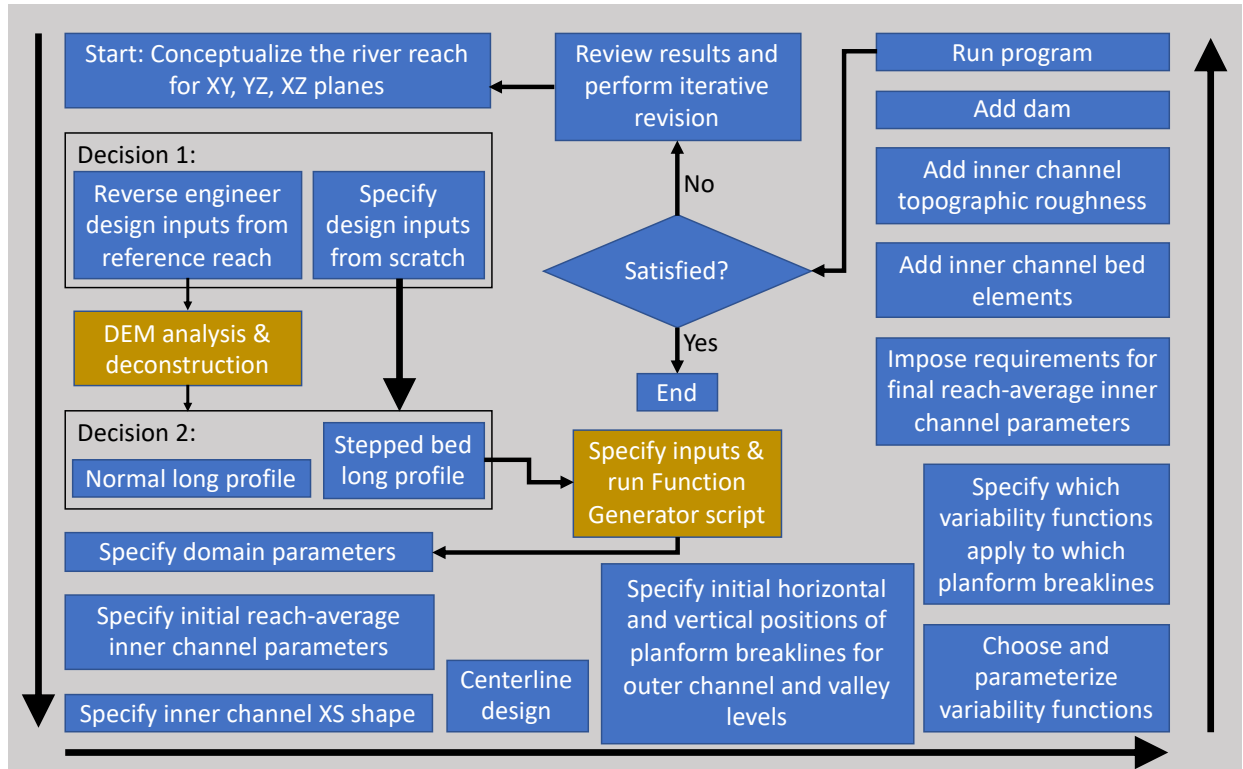


Figure 13. Schematic showing the major steps river valley design using River Builder.

## 4.2 Baking From Scratch Or Mimicking a Reference

River Builder is a wide-open tool to create rivers from scratch. That is very empowering but also very daunting. In practice, the odds that a person can make decisions about what they want for all the complex features of a river going from true scratch is low; people need a conceptual reference of some kind to get started. The real question is just how far will a user want to go with a reference from rough concept to detailed, quantitative mimicry?

Beginning with River Builder 1.1 there are two decision points in the overall workflow. The first decision point addresses your method of obtaining the values for all the design parameters. One option is to specify all of them from scratch based on your concept for what you want. In general, this is going to yield a simpler river design, because it is difficult for people to hold in their mind and work out a large number of features for each element of a design, though it is possible. The other option is to “reverse engineer” design inputs from a real river (this topic is addressed in the next section).

Reverse engineering a river is not only feasible (Figures 14-15), but it can yield simplified geomorphic archetypes with hydraulically and geomorphically equivalent functionality, possibly ecologically equivalent too. This is an active area of research. When using River Builder, you should decide from the outset whether you intend to literally “reverse engineer” an existing real river or if you will simply use data and information from other rivers as a general guide to the design.

Quantitative reverse engineering has always been a topic we have discussed, but until recently it has not mattered much, because we lacked the disciplinary tools to reverse engineer rivers in sufficient detail to parameterize the scope of what River Builder can do, though the underlying math is pretty old and well-known. Now that has changed.

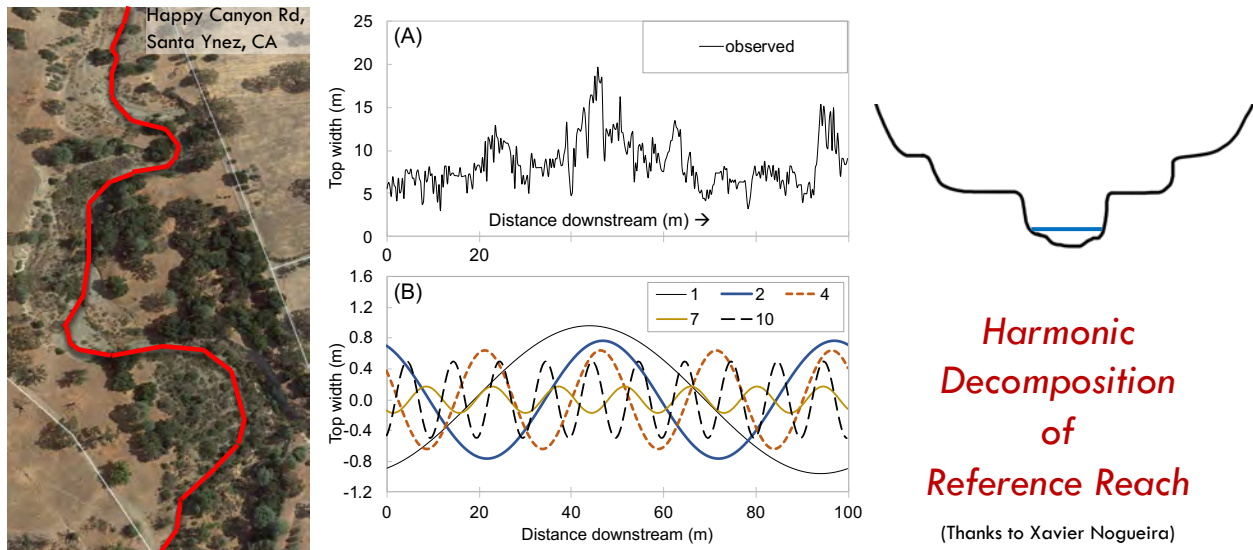
Many researchers around the world are developing scripts to analyze rivers to yield a variety of data that could be suitable as inputs for River Builder, even if they are not intended for that purpose. In the Pasternack lab group, we began creating tools for reverse engineering real rivers in late 2020 on an as-needed basis. River Builder does not currently include a comprehensive suite of tools for reverse engineering, though such tools will likely be added to the “Tools” folder on the River Builder Github as they become available.

To successfully reverse engineer a river to obtain River Builder inputs, two steps are needed. First, one must analyze the river’s DEM to extract all of the information one needs. However, you wouldn’t want that in a generic table format and then have to manually input it yourself somehow, because there is far too much to be able to do that. For example, deconstructing channel width into its constituent geometric parts could require 20-200 individual components (Figure 14), and that’s just one pair of planform breaklines (left and right bank), so multiple that by as many breaklines as desired. When this is done for multiple geometric aspects of a river, you can obtain a high-quality archetypal representation of a channel type without the specific details of an individual site (Figure 15), or the survey and interpolation errors endemic in field mapping.

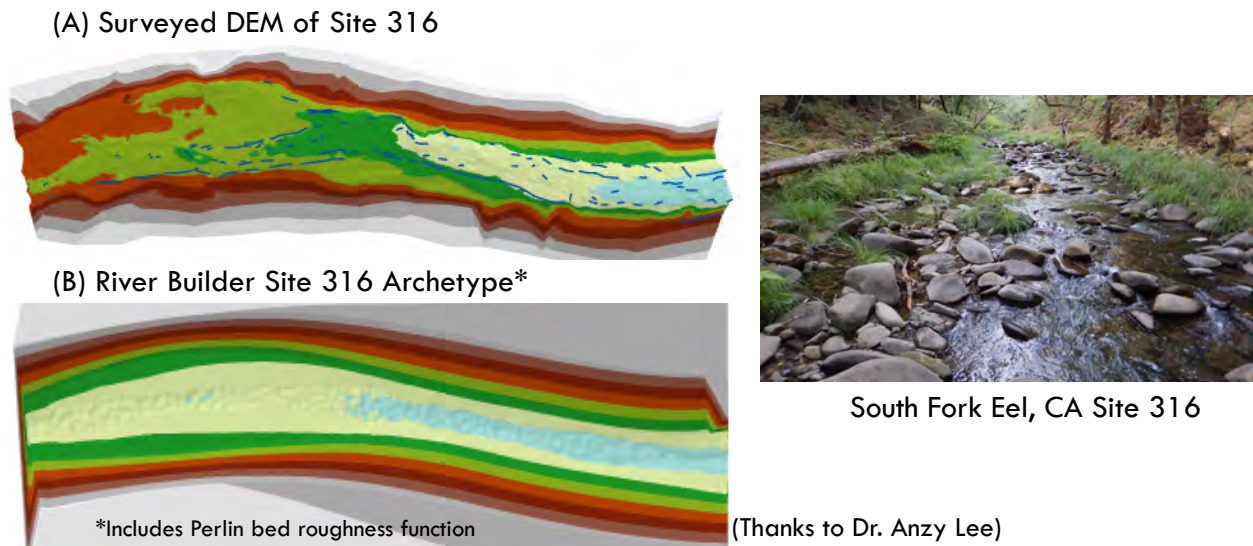
The second step that is needed is a tool that can take the results of a DEM analysis and write them into the proper text format to go into the River Builder input file. This is not necessary difficult to achieve; it is a work in progress. Again though, just how far does one want or need to go to mimic an existing site in order to have the kind of river archetype they want... nobody knows just

yet, but the user should think about this.

If a user desired to design from scratch, then the key is to have conceptual ideas about all of the components needed. At this stage, the GCS options in River Builder are the capabilities for which is most difficult to conceptualize, because there is a lack of GCS studies to learn about what they should be like for rivers in different settings. Thus, the user will have to make those up at their own discretion as of yet.



**Figure 14. Example of reverse engineering geometric functions to use in River Builder. In this case, the bankfull width series from a stream in Santa Ynez, CA was extracted from a Dem created using airborne lidar. Then, the frequency components yielding 95% of the variability in width were extracted, of which 5 are shown.**



**Figure 15. Total reverse engineering example. (A) A site representative of one of California's regional channel classes. (B) A simplified archetype of the field site that captures its**

**essential topographic regime sufficient to replicate hydraulic patterns.**

### **4.3 Stepped Profile Decision**

The second decision point in the overall workflow addresses whether you want to implement a stepped riverbed using the FunctionGenerator.py tool or not; that also includes whether to have a single large waterfalls, if not a long sequence of steps. If steps are not too vertical, then it is very possible that harmonic decomposition can sufficiently represent the longitudinal profile – if you have a reference reach to use for reverse engineering. However, you may want or need to make a stepped bed with a lot more control, so then you would need to plan to use FunctionGenerator.py. This decision is important to make early on, because of the constraints described earlier for this tool.

### **4.4 Domain and Initial Reach-Average Values**

Second, you determine some basic domain parameters and initial reach-average values for the geometric elements of the straight, valley-orthogonal Vanilla River your design will be based on. Domain parameters are the basic set up of the coordinate system and the site dimensions, so they are simple. You might imagine you could specify your final desired reach-average bankfull or inner channel dimensions at the outset, but this is not possible, because at this early stage in design you have not picked and set up your geometric variability functions, so who knows how these variables will be changing down the river. Thus, you are specifying initial values, and then there will be different options about how to handle the effect of variability later in the design process. Key values to specify include reach-average slope along the valley excluding the effect of any channel sinuosity (noting that the valley itself may be sinuous and this slope value is along that sinuous valley centerline), inner channel minimum width, inner channel minimum depth, and then the lateral and vertical offsets for all in-channel and in-valley lateral slope breaks as illustrated in Figure 7. Some variables may be computed from theory and other variables to insure adherence with process concepts, such as computing mean bankfull depth from the Shields equation. Of course, any computation could introduce unit-specific values, so care is needed to insure all values are in the same units throughout all design steps.

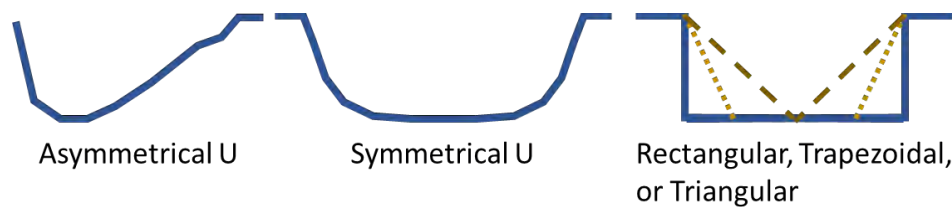
### **4.5 Inner Channel Cross-Sectional Shape**

Third, you have to specify the cross-sectional shape you want to use for the inner channel. Cross-sectional shape is specified as one type for the whole length of the river, but of course it will scale in size and proportion as a result of any changes in inner channel width downriver that you specify. There are three primary cross-sectional shapes you can choose from: symmetrical U-shape, engineered geometric shape, or asymmetrical U-shape (Figure 16). The symmetrical U-shape is the basic shape for a natural river that has low sinuosity. The engineered geometric cross-section option allows a user to create a straight-line trapezoidal shape that can be anything from a triangle to a trapezoid to a rectangle. This would be sensible if the user is defining a canal or other such simple engineered channel shape. Intermediate trapezoidal shapes with any side-slope are possible.

The asymmetrical U-shape shifts the deepest part of the cross section back and forth across the river to stay at the outside of the meander bend on the basis of how local channel curvature compares to the global maximum curvature. A challenging issue with the asymmetrical U-shape option is the decision as to what length of channel to use to compute a “local maximum curvature” that theoretically governs a channel’s cross-sectional asymmetry. In traditional geomorphic theory



and presentation of this concept, the idealized centerline takes on a simple sine function shape, so it is trivially obvious what channel length and curvature calculation method to use to have the deepest part of the cross-section at the outer bank. In that case, local and global maximum curvature are the same and there is no technical problem. However, more realistic centerlines have numerous undulations of varying amplitude, phase, and frequency, so it is extremely difficult to set an objective criterion for what length to use to compute a reference local maximum curvature. Further, River Builder allows for looped meandering (aka “highly curved”), as discussed shortly, so there can be very high curvature, and that can happen at any scale depending on how many nested variability functions a user builds into the complexity of their meandering centerline. The procedure for how River Builder specifies centerline curvature is explained later; for now the important point is to understand that channel asymmetry is governed by that curvature. However, if you are dissatisfied with the longitudinal positioning of your asymmetric pools and riffles, then you may wish to use a lot of care in specifying the centerline function or switch to the symmetrical U shape option.



**Figure 16. River Builder’s inner channel cross-sectional shape options.**

## 4.6 Centerline Design

Centerline design refers to the initial concept of the nested sinuosities of the channel and valley (as covered in section 3.1.1). Of course, once asymmetric functions are applied to the left and right bank functions, then this conceptual centerline may no longer be the actual centerline of the inner channel. Similarly, the “thalweg” may not occur where the centerline is either conceptualized or actually present, because the inner channel cross-section shape may be asymmetric, too. One may reverse engineer the centerline function from a real site’s data or design it from scratch.

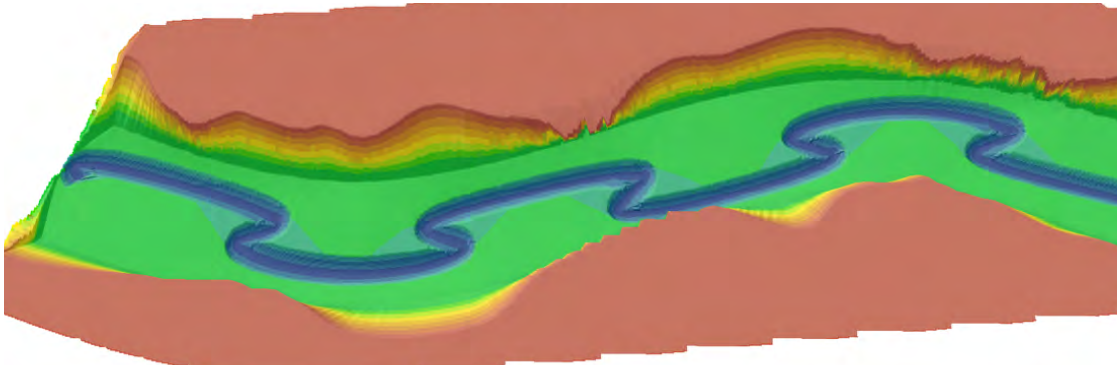
## 4.7 Planform Breaklines

Fourth, you formally select all the lateral slope breaks you want to use in your channel and valley zones (e.g., Figure 7). For each one, you must specify the horizontal and vertical offsets from the accordingly centerline. Left and right slope breaks are independent, so they do not need to have the same offset values. There is an enormous amount of freedom and capability possible with this aspect of the program. Negative vertical offsets are permitted, but negative horizontal offsets are not permitted. A negative horizontal offset is no different than specifying another contour closer to the centerline, so it does not make sense to allow that; if you want a contour closer to the centerline, then simply specify that before the next one out. Note that, since all breaklines parameters are refer to centerline, there is no easy way to check beforehand whether breaklines are overlapped with each other or not, and this issue will leave with user to solve. Ideally, with breakline layer increases, its minimum lateral offset and height offset also increase.

## 4.8 Geometric Variability Functions

### 4.8.1 General concept

Fifth, you turn the slope break points from your cross-sectional conceptualization into plan-view breaklines (aka contours), including specification of the geometric variability functions for each contour (Figure 20). Brown et al. (2014) only presented additive sinusoidal variability functions, but River Builder now has several more functions available. The functions include linear, sine, cosine, sine squared, cosine squared, Cnoidal, square wave (i.e. flat top and bottom with vertical risers), three-parameter Perlin noise, and gooseneck curve that loop back on themselves (Figure 17).



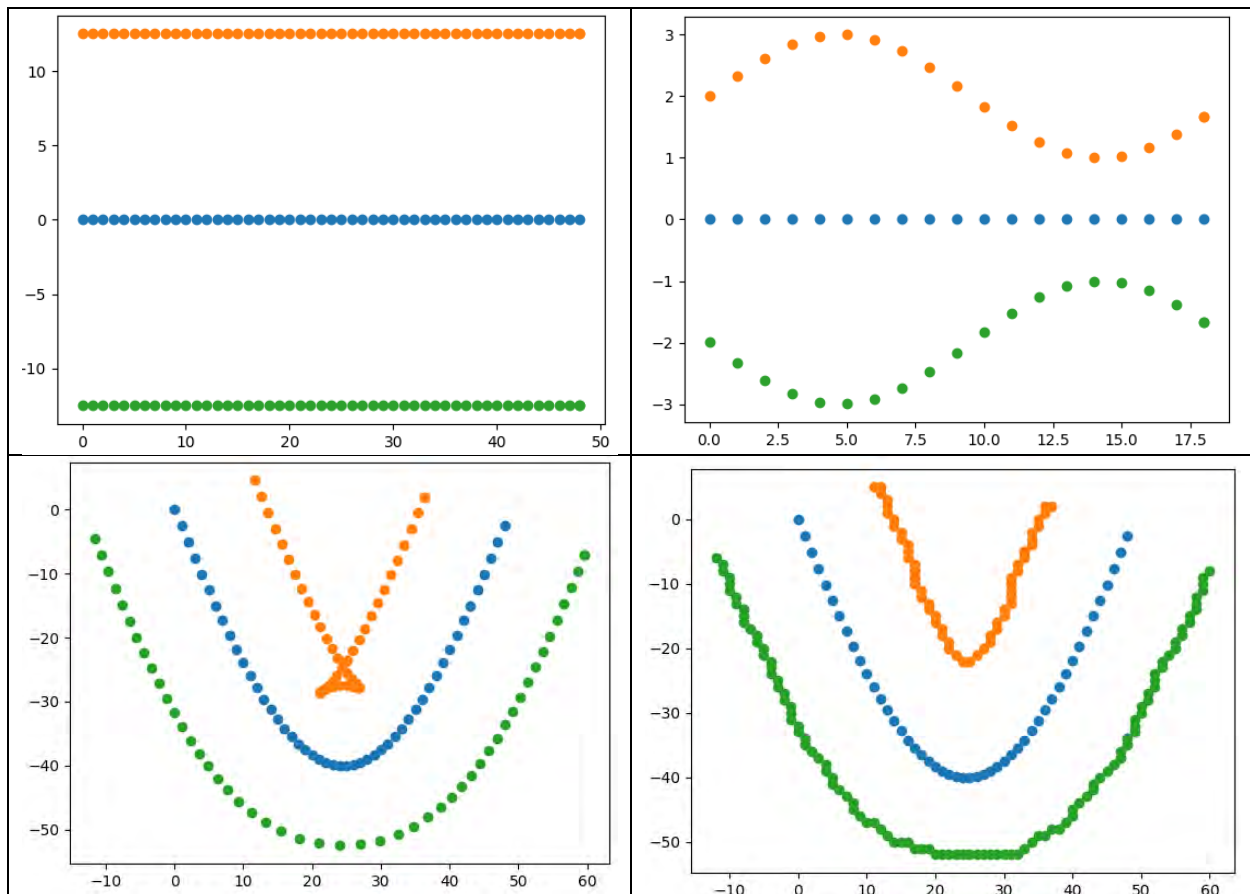
**Figure 17. Example of a steep-sided sinuous river valley with an inset gooseneck-meandering channel. Obviously these are tighter looped curves than natural, but they certainly emphasize what is possible with the software.**

The decision of which function or combination of functions to use for a given breakline rests on a firm geomorphic understanding of the local reach, its river, and the region. For each breakline you can add together as many individual functions as you want to define a more complex geometric variability function. For example, let's say a river is widening downstream but also oscillating its width. You can design that by adding a line function and a sine function. Or as another example, let's say a real bed longitudinal profile includes many frequencies of undulation. You could perform a harmonic analysis to obtain the parameters for all the sinusoidal functions necessary to replicate the periodic components of the natural signal. Then, you could mimic the non-periodic residual by adding a Perlin noise function.

### 4.8.2 Automated repair of contour self-crossing

Now that you understand that you can specify whatever lateral variability you want to each breakline, then consider the consequences of what will happen to that breakline when not only its function is implemented, but it is implemented on top of both channel and valley centerline variability you specify. Figure 18 illustrates the process. A user begins by specifying a centerline and its inner channel width (upper left panel). Then a width variability function is applied (upper right panel). Then centerline meandering is applied to that undulating channel, causing so much bending that the inner bank contour now crosses over itself (lower left panel). In a real-world sense, the contour points that are now inside the channel are no longer necessary and can be removed. Through some nifty coding, River Builder is able to identify these cross-overs and eliminate the extraneous points (lower right panel).

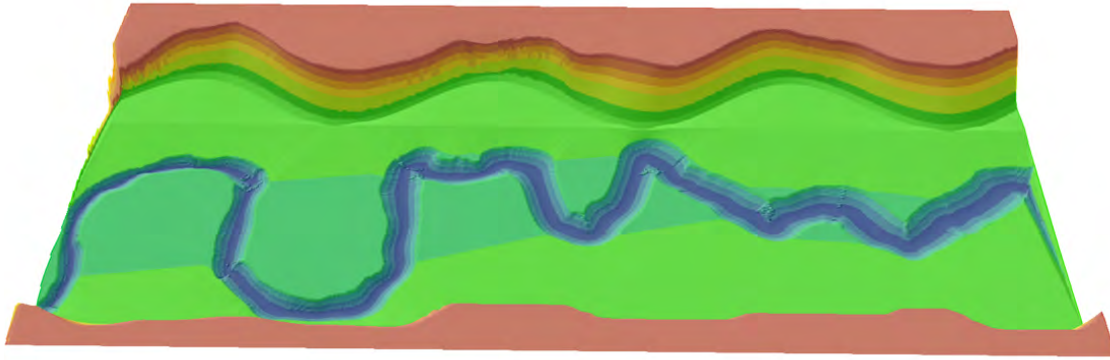
Note that this only applies for the situation when a contour crosses itself. As discussed in section 3.1.2, because of the specification of lateral offsets now being relative to the centerline in version 1.2, you may end up with two or more contours crossing each other. River Builder is not capable of automatically detecting and resolving that different problem, the way it could prevent it in versions 1.0 and 1.1. This has advantages and disadvantages, but in any case this is the current situation. Therefore, you can count on River Builder to insure no self-crossing of a contour no matter how much you bend it with complex and additive geomorphic variability functions, but not to address crossings of different contours.



**Figure 18.** Illustration of the automated repair of twice-bent contours. Upper left shows an initial vanilla channel. Upper right shows a width variability function applied to that. Lower left shows what happens when that is then bent to apply centerline meandering, yielding unacceptable contour cross-overs. Lower right shows the outcome of the automated repair in which River Builder deletes out the unnecessary points and solves the problem.

#### 4.8.3 Piecewise varying functions

River Builder for Python is capable of changing geometric functions at user-designated positions down the valley. This is termed “piecewise varying”. This is possible for any breakline. Conceptually, this is done by specifying the length of the river along which each geometric variability function applies. This step involves applying a “mask”. How this is done will be explained later.

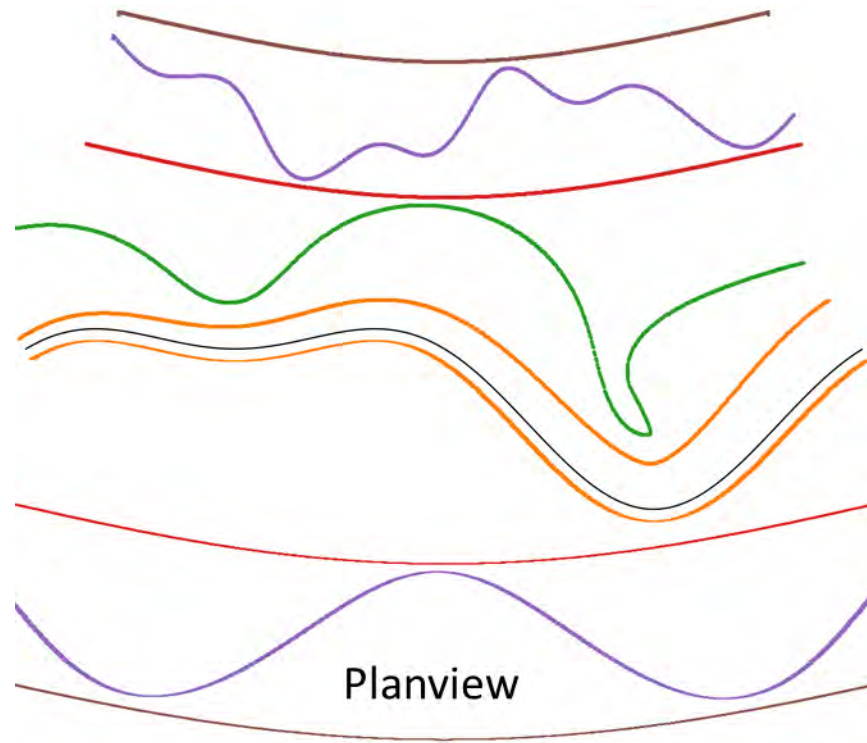


**Figure 19. Example of a channel whose centerline function varies at user-specified positions down the valley (i.e. a piecewise varying centerline). From left, it begins with a gooseneck meander function and then switches to other types down valley.**

Once you select all your functions, you have to set the parameter values for each one. All functions have parameters. For example, lines have slopes and y-intercepts. Sinusoidal functions have amplitudes, frequencies, and phases. You have to decide what values to use for all the parameters. Further, you can use the same function for multiple breaklines, but if you want to change the parameter values, then that essentially makes it a new instance of the function.

Although parameters for each equation may be specified independently, self-maintenance processes in rivers often produce coherent patterns among multiple geometric elements, including mutual longitudinal variations identified through spectral or wavelet analysis of spatial series or the GCS function between them. Thus, the key decision for parameterization is to decide which two or more variables will be linked to vary coherently. For that to happen the variables have to be described by the same equation, and then parameters governing frequency, amplitude, and phase are specified to yield oscillations that produce the desired landform structure. This is where GCSs come into play. Beyond creating landforms, the deeper goal is to obtain topographically steered, stage-dependent hydraulics that drive a variety of channel maintenance mechanisms when operated on by a natural flow regime. At the same time, there is another goal of providing spatially organized yet heterogeneous aquatic and riparian habitats to serve different species needs in their various life stages.

Remember, geometric functions are defined relative to the stream-wise and valley-wise coordinate systems. Therefore, in your mind's eye you have to think of them that way. It is possible and common that the nesting can get so complicated that contours will cross over. River Builder will erase any such crossovers and insure a smooth topography.



**Figure 20. Example of the plan-view result of projecting slope break points into plan-view contours. Black line is centerline. Orange lines are inner channel banks. Green line is an additional slope break that could be viewed as in the channel or on the floodplain depending on the conceptualization a designer has. Red lines are the slope breaks from a macro channel to the valley floor or potentially from the floodplain to adjacent terraces. Purple lines are additional floodplain or terrace slope breaks. Brown lines are the outer-most valley boundaries of the synthetic river valley in this design.**

#### 4.9 Apply Functions To Breaklines

Because you may wish to apply the same geometric variability function to multiple channel and valley breaklines, the step to define those functions is independent of the step to apply them to one or more breaklines. As a result the sixth step is to select the functions you want to use for each breakline and write them out. River Builder enables you to add together as many functions as you want for each breakline. Just remember that you may need unique parameter values for different breaklines, so you may have to create a unique set of functions for each breakline if you want that to happen.

#### 4.10 Parameter Optimization

Users specify initial values for river variables such as depth and width, but once all the variability functions and objects are added, the reach-average values of these variables may be quite different from those initial values. Theoretically, one could do the math to get all this right, but that is very hard if the design is complex. Most likely, some variables are particularly important to a user compared to others. As a first check, River Builder produces a text output file with several reach-average values, so you can check to see how they turned out. Then, one could manually iterate



inputs to try to mindfully yield the outcome you want. However, that would be tedious. Therefore, River Builder includes an optimization routine that allows the user to specify key variables to hold constant at set values, but then there will be freedom to adjust other parameters or variables to achieve the desired outcome.

#### 4.11 Add Objects

The last design steps involve specifying channel bed elements, bed roughness, and the number and locations of any dams.

#### 4.12 Run River Builder

Once the text input file is set up you run River Builder to render the 3D terrain and output results. The most important output is a .csv point file that has the coordinates of all the points in the terrain. To help with iterative design and to check design characteristics against expectations, several other outputs are provided, including data tables and 2D plots. River Builder does not have a 3D plotting capability, so you have to import the final .csv point file into another program to visualize it in 3D. Based on analysis of all outputs, you may choose to iterate the design to refine and improve it until you have the desired outcome. It is recommended you keep your original files for each attempt and use a master log file to document and track all your attempts. This will preserve your design process in case you want to go back later to review what you did or return to an earlier design instance.

### 5 River Builder 1.2.0 Highlights

As in the past, **no knowledge of coding is required** to use this software. The user interface is as simple as it can be (i.e. a text editing program of your choice and a simple text input file). To run the input file, you do have to have Python3 on your computer along with two packages: python3 numpy and matplotlib; all of these are free. Note that all required packages come with ArcPro, so if your Python3 installation came with ArcPro, then you are all set. Otherwise, just download those packages (see steps in section 6.1 below) and you're ready to go. Also, this manual tells you the one and only command you have to run on your computer to make the program evaluate your input file, so that is very accessible for everybody to do. We also now have a pre-established script you can edit to run the program instead of doing it right from the command line.

In considering how to proceed with development of River Builder from the previously existing R version (0.1.1), several factors came into consideration. First, we re-evaluated the future of the R platform relative to the opportunities and challenges we see in the future. A key consideration was the fact that we have also developed a companion “River Architect” software platform in Python to analyze both natural and designed river valleys. More of our coding is being done in Python than R.

Second, we came to the realization that we needed to revisit and further advance the equations in Brown et al. (2014) to open the possibilities for many of the new features in this version of River Builder. Science is an iterative endeavor. We have learned a lot through time, so it was now necessary to start over from scratch and implement a next-generation code.

Third, we needed to significantly improve the structure of the code to institute best practices in coding.

Based on these considerations, we came to the conclusion that River Builder would have the greatest potential for use on many devices and in many applications if programmed in Python instead of R. Thus, we chose to re-write River Builder from scratch and implement it in Python3. We chose Python3 over Python2 in light of the pending retirement of ArcMap that uses Python2. ArcPro uses Python3. At present River Builder does not use any “arcpy” functionality, but programming in Python3 preserved the option to do so in the future.

The overhaul of the code has led to several major new features and fixes, as summarized next. This list compared River Builder 1.2 in Python vs River Builder 0.1 in R.

## 5.1 Major New Features

- Different channel and valley centerlines, with channel centerline and associated coordinate system nested inside valley coordinate system (Figure 4; Figure 20).
- Channel now broken into independent left and right sizes, with overbank valley region still having independent left and right sides (Figure 7).
- Dedicated “inner channel” that is assigned a formal XS, using all the same shape options as before, and they all work correctly now, including asymmetric U (Figure 7).
- Outside the inner channel, outer bank and valley zones each now have an unlimited number of XS slope breaks, with each break point being assigned a vertical and horizontal offset from the channel centerline (Figure 7).
- Each outer channel and valley breakpoint may be assigned positive, zero, or negative offsets, which enables users to create levees, roads, and yazoo channels (Figure 7).
- New geometric functions have been added. These may be used for any channel or valley feature that may use a variability function.
  - The Perlin function now has a higher order functionality with octaves to enable more fractal diversity.
  - The Cnoidal function is now available to allow for creating a periodic function with longer, flatter zones separated by more peaked zones as compared to the sinusoid function. Parameters control how long and flat the flat zones are.
  - The nearly rectangular “step wave” function has been added to allow flat sections separated by a nearly straight, perpendicular line on the basis of a representation using many cosine functions
  - The  $\sin^2$  and  $\cos^2$  functions have been added. Line, sine, and cosine functions remain. The value of the squared functions lies in their ability to produce two positive peaks over the range for which the sine and cosine functions only produce one positive peak.
- Unlimited number of geometric functions may be added/subtracted together to make a single complex function with multiple features with different frequencies.
- A looped (aka “highly curved”) meandering function has been added for use with the centerline (Figure 17). This allows for multiple Y values for each X centerline position, which is what is needed to create “gooseneck” meandering.

- New 2D plots to help users evaluate renders, including independent plots for whole valley XS vs just channel XS.
- Initial reach-average values a user specifies only hold true before a user adds geometric function variability. These values are now minimums that variability is added on to. The code now computes the actual final reach-average values of channel slope, width, and depth, so a user can decide whether to go with those values or iterate the design. We also added an optimization routine to allow user to enforce final desired channel-wise reach-average metrics like width and depth by enabling program to change geometric function parameters as needed to end up with the right values.
- Optimization routine so that a user may specify the final desired values for some parameters and then adjust specified variability function parameters to guarantee those final outcomes.
- A mask functionality now applies to all functions. A user can specify at what ranges of distance along the channel or valley centerline a function should turn on, and what ranges of distance the function should turn off. This provides a greater flexibility for users to create a non-periodic curve (Figure 19).
- A “smooth” parameter used to smoothen the centerline. It is extremely useful to prevent sharp changes and to create a more natural channel shape when piece-wise functions are applied to centerline.
- A new way to specify functions. Instead of listed all the functions directly in the input file, a user can store those functions into a txt file, and the program will read them out from it.
- Users are able to add various bed elements (depressions and protrusions) to the inner channel bed..
- Users are able to apply a 2D Perlin fluctuation bed roughness to inner channel bed.
- Users are able to add dams to the inner channel.
- River Builder now provides a tool to generate functions needed to build a step pool thalweg based on user specifications.
- River Builder now comes with a Python script to run the program, “rb\_run.py”.
- We have a new script you can use to quickly build 3D terrains in ArcGIS from your River Builder outputs. It is called “rb\_to\_terrain.py”.

## 5.2 Major Fixes

- The new code is natively, entirely in stream-wise coordinates, so everything that is done is in line with or perpendicular to the centerline now. This applies to both the channel and valley centerlines.
- The new code has a routine that prevents topographic contours specified with geometric functions from crossing over. When crossovers would occur, the code deletes crossed sections and smoothly interpolates a good surface to avoid any problems.
- The previous equation for asymmetric U cross-section was based on x-y coordinates and calculated global curvature in such a way that the function could break if attempted to generalize for all possible river shapes. In theory the code should use some sort of local curvature for complex centerline meandering. However, local curvature is tricky, because you have to decide what counts as local; if you choose too small a scale, every random fluctuation can become a crazy local maximum and channel asymmetry can go wild with too many undulations back and forth. The new code uses an advanced algorithm to compute curvature at each point that is natively following stream-wise coordinates. This new

calculation is determined only by how the river bends, no longer affected by the position of the river. Specifically, it looks from one point to the next to create a vector between those points. Then it computes the angle between any two such adjacent vectors. The maximum angle sets the global maximum curvature. The degree of cross-section asymmetry is only at this maximum level when a meander bend is as sharp as the maximum curvature location. For bends with lower curvatures, the asymmetry is scaled linearly lower. As long as meandering is well-behaved, then the function will work well, but if extreme high-frequency noise is added along the meandering centerline, then it is possible that the maximum curvature value may not be very meaningful for controlling asymmetry at the smoothed meandering scale where it belongs. In such a situation, there is an advanced feature where the user may specify their own curvature function instead of using the internally computed one. This is explained in section 10.6.

- Addition of crash protection features so that if a user specified impossible values or no values, then the code defaults to set default value and the code still runs. The program outputs a warning message and shows all the values used, so a user may figure out where the problem went wrong, but they still have a working design to study.
- Deletion of all geomorphic covariance structure (GCS) computation for now, because it was being done in the straight valley coordinate system, not the stream-wise system. Will add this back later as a feature addition when we can program it properly.
- A more sophisticated way to calculate the elevations of breakline points. Now the elevation of breakline points is calculated from its closest centerline thalweg points.
- A more sophisticated way to calculate river slope. When the river runs in the same direction with the valley, the river slope should be the same as the valley slope; when the river runs perpendicular to the valley (because of sinuosity of the river), the river slope should be much smaller than the valley slope. Based on this observation, a local river slope is calculated for the centerline; it is determined not only by the terrain slope, but also by the local sinuosity.
- Beginning in version 1.2, changed lateral and vertical referencing from incremental distances between breaklines to whole distances from the centerline.
- Although not a “fix”, there was a major change to the way topographic points get output to a text file for use in other software. Specifically, the addition of bed elements, bed roughness, and dams to the inner channel necessitated use of a square point grid so that these features can be resolved. Previously, points were output along the centerline, inner channel contour lines, and breaklines beyond the inner channel. That was no longer viable once we implemented these new features. The way this works now is explained in the description of the “X Resolution” parameter.

### 5.3 Major Features Removed

- The Caamano criterion for two-stage flow convergence routing (aka “velocity reversal”) was previously computed for fixed, assumed riffle/pool locations for a few example cases with a straight channel. Thus, the old outputs would be wrong for most cases. This feature has been removed until it can be re-programmed to compute for each sequential riffle-pool couplet.

## 6 SOFTWARE DOWNLOAD, INSTALLATION, AND SET-UP

### 6.1 Install Python or Locate Your Existing Copy

Use of River Builder does not require any programming skill, but you do have to install the free software, Python in version 3.x, where x is any version number within the 3 series. Python 3.x is available for free for the Windows, Mac, and Linux operating systems.

If you use ArcPro on Windows, then you already have Python 3.x on your computer and the two packages, python3 numpy and matplotlib. It is recommended that you use this Python program; do not re-install another version elsewhere on your computer. An example of where you might find this program in ArcPro could be something like:

```
C:\Program Files\ArcGIS\Pro\bin\Python\Scripts\propy.bat
```

If you do not use ArcPro, then simply go to the Python website and download Python 3.x to your computer using the link below. Follow its instructions to install it.

<https://www.python.org/>

You can install numpy and matplotlib by typing the following commands into your windows command line. To access the command line, go to your Windows search bar and type “cmd”. That will bring it up.

```
PATH\TO\PYTHON.EXE -m pip install numpy  
PATH\TO\PYTHON.EXE -m pip install matplotlib
```

In the two commands above, you have to substitute your file path for the “PATH\TO\” part of each command. If you are not sure where your Python executable file is located on your computer, then you can run a search for it in Windows Explorer. The -m option means run the module as a program.

### 6.2 Select Your Preferred Text File Editor

River Builder works by having you set up your design in a single text file. Thus, you need a text file editor (aka word processing program), which is available for free on all computer operating systems (e.g., Notepad on Windows and Textedit on MacOS) and from many cloud service providers. You can use any text editor you want- no special functionality is needed.

The input file for River builder uses the “.txt” file type. Be sure that when you finish preparing your text file in your preferred text editor that you save it back to this file type.

### 6.3 Download River Builder Python Code

River Builder 1.2.0 is publicly available for free as a download from the Github repository at the link below. The Github repository offers the essential Python package, many worked sample files, and any add-on tools we have programmed to help users develop special cases for which designing the variability functions can get very complicated.



<https://github.com/RiverBuilder/RiverBuilder>

In addition to the Github repository, there is a dedicated website that supports training and research with River Builder at the link below. Over time, training videos, more worked samples, workflows, and other aids that are not suitable for storage on Github will be available here.

<https://riverbuilder.ucdavis.edu>.

We will do our best to release updates to improve the existing software and build in new capacities over time. When we do, they will be available at these two sites. We do not have a newsletter or anything, as we are just a couple of academics, so you just have to check the websites periodically.

## 7 RUNNING THE PROGRAM

We recommend you rename the “river builder” folder to the simpler name “RBpy” just to make it easier to type commands with a shorter name.

Place your text input file into this RBpy folder. It can have whatever name you want to call it. A good nomenclature to use is to add a numerical suffix to the name so that as you iterate the design you can just increment that number. For the purpose of this manual, we will use the file name “NAME\_001.txt” as the example. NAME would be replaced with your preferred design name and then as you try things you would increment from 001 to 002, 003, etc.

You can have as many input files as you want in the folder.

### 7.1 Windows OS Simple Approach

You can run River Builder with a single instruction that is easy to write and implement. This command is implemented from within the RBpy folder. The command has the following structure:

```
"A" -m riverbuilder B.txt C "D"
```

where “A” represents the file location for your Python 3.x executable file, riverbuilder is the River Builder Python package, B.txt represents your text input file name, C represents the name of the new folder within RBpy where you want the program to place your results, and “D” is a simple text comment you can add to the log file, if you want to. C and D are not required. If C is not specified, the output folder will be “riverbuilder\_output”. Please put the comment D in quotation marks, if you choose to use it.

An example of such a single command is provided below:

```
"C:\Program Files\ArcGIS\Pro\bin\Python\Scripts\propy" -m riverbuilder NAME_001.txt  
NAME_001output "a simple test."
```

Note that in the above example, the ArcPro version of Python is being called, which has the

filename “propy”.

To implement such a command, follow these steps:

Open a Command prompt window by clicking on the search bar and typing "cmd".

In the Command prompt window, navigate through the file directory and into the river builder folder, now called "RBpy".

In the Command prompt window, type your single command to run the program, such as:

```
"C:\Program Files\ArcGIS\Pro\bin\Python\Scripts\propy" -m riverbuilder NAME_001.txt  
NAME_001output "test1"
```

That's it. The code will run and your results will be produced.

The above example is generic and cannot be used specifically, so here is a sample command to actually run River Builder using the S1.txt input file provided to you in samples\S1 folder on the River Builder Github.

```
"C:\Program Files\ArcGIS\Pro\bin\Python\Scripts\propy" -m riverbuilder  
samples\S1\S1.txt samples\S1\S1 "a simple test"
```

The samples folder has many worked examples that illustrate different kinds of river scenarios. Examples S1-S6 came from the Brown et al. (2014) article on Synthetic River Valleys. Today, the outputs may not look like they did in that article, because as River Builder has advanced there have been changes to various features that yield more realistic outcomes- especially for non-orthogonal valleys and channels. Nevertheless, it is very helpful if you work the examples to gain an understanding of how to use the software.

## 7.2 Python Integration

To enhance the ability for users to integrate River Builder with other python tools, River Builder can also be used as a regular python package.

Navigate to folder containing the riverbuilder folder and you will see a script there called “rb\_run.py”.

Open rb\_run.py with a text editor and write down the following code lines:

```
from riverbuilder.core import river  
  
fname = ".\samples\S1\S1.txt"  
outfolder = ".\samples\S1\S1"  
log = outfolder + "\S1_log.txt"  
  
# Run RiverBuilder  
river.buildRiver(fname, outfolder, log)
```

In the Command prompt window, type the following line to run the python script:

```
"C:\Program Files\ArcGIS\Pro\bin\Python\Scripts\propy" rb_run.py
```

When this rb\_run.py file is run, the code will run and results will be produced in samples\S1\S1 folder.

## 8 HOW TO USE RIVER BUILDER

The essence of using River Builder comes down to simply entering your design values into the input text file and then running the one command in the previous section. Its that simple... and yet it is also quite challenging, because you have to know what values you want to put into the input file. The software does not tell you how to specify a river design, but by offering you a set of design options, it certainly focuses your effort on the key components.

The default input text file has pre-defined, acceptable minimum values for the program to function correctly. For full assurance, the user must follow the guidelines specified at the top of the text file.

## 9 PROGRAM OUTPUTS

As soon as River Builder stops running and the Command prompt window returns to its default ready status, you will have a new folder within RBpy having the name you assigned in your command to run the program. In that folder there are a number of files, which are described next.

### Log.txt

Contains a listing of all the parameters and functions used to run the model based on your input file. It also includes “Alert!” messages that tell you how the program handled any missing information, usually by assigning a default value or turning a feature off.

In many instances the “Alert!” message is not an indicator of a problem of any kind; it’s just there to encourage you to affirm that the decisions you made were what you intended. However, always read the messages in case there is something you meant to specify and did not.

### SRVcenterline.csv

A comma-limited text file containing the bed slope and planform curvature along the centerline. This can be used for using and plotting these profile data in another program.

### SRVcurvature.png

Displays plots of the bed slope and planform curvature series from the SRVcenterline.csv file.

### SRVinnerChannelXShape.png

A plot of the cross-section of the inner channel. There will be two plots if an asymmetrical cross-section is used, with one at the curvature closest to zero and one at maximum curvature. If symmetric U or trapezoid, or if no curvature changes, there will be only one plot. The legend shows the X position(s) of the cross-section(s)

#### SRVlevels\_xy.csv

A comma-limited text file containing the planform points for all the specified contours

#### SRVlevels\_xy.png

Planform plot of the river showing the positioning of the river's features.

#### SRVlevels\_xz.csv

A comma-limited text file containing the elevation longitudinal profiles of all the contours, including the thalweg.

#### SRVlevels\_xz.png

Longitudinal plot of the river showing the elevation profiles for all specified contours

#### SRVmetrics.txt

This file contains statistical analysis results from analyzing the river after it has been rendered with the variability functions. Computed values can be used to iterate the design as needed to get what you want it to be given a complex subreach meandering function. River Builder does have an optimization routine to help with that, but sometimes you may want to adjust different parameters than the ones selected for optimization. The information can be organized into two categories- channel and valley data.

- Slope & Sinuosity: The metrics file tells you the actual channel and valley slopes and sinuosities (along the centerline). Recall that the user enters a straight valley slope in the input file, not the final sinuous valley slope, not nested sinuous channel slope. Only for a straight channel will the valley and channel slopes be the same.
- Average Depth and Width statistics: mean cross-sectional depth (H) and top width (W) values are computed along the channel, perpendicular to the channel. Also, the coefficient of variation of each is computed for the whole reach.

#### SRVtopo.csv

This is the primary output that is the purpose for River Builder. This file contains all of the comma-delimited XYZ coordinates for the synthetic river valley. Points are provided along the user-specified contours that define the river corridor as well as along a user-selected number of intervals in the inner channel. The first row of the file contains the header, {X, Y, Z, Label}. Label is a 2-letter designation for what contour a given set of points represents.

#### SRVvalleyXSshape.png

A cross sectional plot that spans the entire valley width.

## **10 RIVER BUILDER INPUTS**

River Builder works by having the user write all of the information to create an artificial river corridor in a single text file. It cannot get any simpler than this, which is why we chose this approach. The text file is written in plain English that is easily readable, but for the specific input lines you do have to follow the required syntax.

Here are some pointers about parameters and inputs:

- Every line starting with '#' will be ignored.
- All dimensional numbers are in units of meters.
- User-defined functions may be used for sub-reach variability parameters only.
- Every input with an "=" sign should not have any spaces before or after the "=" sign.
  - For example, a line should be "Length=420" not "Length = 420".
- Bankfull depth can either be (A) user-defined or (B) calculated from the Critical Shields Stress and Median Sediment Size.
- Centerline Curvature can either be (A) user-defined or (B) calculate from centerline slope.
- Calculations of banks of channel are based on channel centerline; calculations of levels of valley are based on valley centerline.
- When providing an input involving pi, do so in the form of A\*pi, where A is a constant. (EX: 2\*pi, pi, pi/6, 3+pi)

## 10.1 Domain Parameters

- **Datum** – a fixed starting point of operation to measure changes of a specific property. Used for thalweg elevation. Input must be a real number. This datum is located at the downstream end of the synthetic river valley, and then the river goes up from there. Thus, if the datum is set as any number >0, then it is guaranteed that all elevations in the valley will also be > 0. You may set any arbitrary number you wish. If you want to estimate the total elevation range of the thalweg along the river, then simply multiply your Valley Slope input by the Length input and that will give you the overall elevation range of the thalweg. For example, for a 1000 m long channel and a Valley slope of 0.01, then thalweg elevation range is going to span 10 m. The default value is 10.
- **Length** – the length of the x-axis of the river valley. Input must be a positive real number. This is only the channel length if the channel and valley are both straight. If the channel is sinuous and valley straight, then this length is the valley length and the actual channel length will depend on the sinuosity. Channel and valley sinuosity are computed by the model and reported in the Data.csv file, so you can compute the final channel length if you want to know it. The default value is 1000.
- **X Resolution** – This parameter controls the planform grid resolution of topographic point output from River Builder. The topographic file with the main output is SRVtopo.csv. The points in this file are on a square grid. The X and Y spacing of the square grid are equal by definition. Therefore, the more sinuous your breaklines and centerlines are, the higher resolution you might want to use. Further, if you have a lot of objects in the river (e.g., bed elements, bed roughness, or dams,), then you would want to use a high value to insure these are accurately resolved in the output point file. Finally, if you have steep banks, you would want a high resolution to help localize those steep features. Note that the point output will not be every grid point in the entire topographic domain. Instead, River Builder will output grid points for the centerline, the entire inner channel (required to show all the bed features), and along all outer channel and valley breaklines. A value of 1 means 1-unit resolution (e.g.,



1 m if you want units of meters). Input must be a positive number. There is no limit to how fine the point spacing can be, other than computation time for your computer. The smaller this number, the longer the code will take to run. The default value is 1.

## 10.2 Dimensionless Parameters

- **Valley Slope (Sv)** – the slope of the valley floor along the X-Z plane assuming a straight valley. It is used to calculate the overall channel slope with a given sinuosity and govern the incline/decline of the surrounding valley. Input must be a real number. Note that during the process of designing a channel it is often helpful to set this to zero, so you can visualize the “detrended” geometry of the river. Once satisfied, then add the desired slope. The default value is 0.001.
- **Critical Shields stress ( $\tau^*$ 50)** – This is an optional input. To use it, set the inner channel depth minimum to 0 and then this will be activated. If you want to use the classic equation to establish a bankfull depth to just yield the shear stress needed to mobilize the bed for a specified bed material grain size, then you can use this by making the inner channel take on bankfull dimensions. Common values used for this parameter are 0.03, 0.045, 0.047, or 0.06, depending on what size fraction you want to insure would be mobilized at bankfull flow. The equation that uses this input is Eq. (4) in Brown et al. (2015). Input must be a positive real number. ***This is the one place where the SI unit system matters in River Builder. If you do not use this function, then there really are no units to your numbers for all practical considerations. The default value is 0.06***

## 10.3 Channel Parameters

The parameters in this group specify initial values for the width and depth of the inner channel (Figure 7). Because width and depth can be made to vary down the river, it is impossible to pre-specify final average values. A choice has to be made as to how to vary these relative to the start values. In this software, the decision was made to have the user set the **minimum** values for width and depth at this stage. If no variability functions are used, then these are the final values. If variability functions are used, then the software computes the average values along the whole channel and reports that to the user. The user can then decide to manually adjust the initial values or the variability functions to get a different outcome, or they can use the software’s optimization routine to force desired average values at the expense of changing variability function parameters. This functionality will be explained later.

- **Inner Channel Lateral Offset Minimum** – the minimum distance between left and right inner channel bank tops on the X-Y plane, not counting any sub-reach variability functions. Input must be a positive real number. The value that the user provides is divided by two and then the result is applied to each side of the inner channel, resulting in an initially symmetrical minimum distance from channel centerline to left and right bank. This variable is specified to protect the user from making a mistake with sub-reach variability functions in which the banks would cross over, yielding an impossible outcome. This insures that the river never has a zero or negative width. If you are confident in what you are doing, then you can set this to zero. The default value is 10.

- **Inner Channel Depth Minimum** – the height offset between the bank top and the highest point of the centerline on the X-Z plane. This prevents sub-reach variability functions from making the depth any smaller than this. Depending on channel slope and variability functions, local inner channel depth can differ from this value. Input must be a positive real number or zero. If zero, then the code calculates an inner channel depth minimum using the critical Shields stress equation in section 10.2. This is intended for creating an inner channel that is the bankfull channel. The default value is 0.
- **Median Sediment Size (D50)** – the particle size of the material comprising the river bed. Input must be a positive real number. This is only used if the user has set the inner channel depth minimum to 0 to compute the bankfull depth value associated with the initialization of bed material transport for this specified grain size. This value is used in conjunction with the critical Shields stress value as explained in section 10.2. Otherwise it has no utility in River Builder. Remember, River Builder creates topography, not sediment facies. Sediment facies are too fine scale for use in the current intended applications for River Builder. The default value is 0.01.

## 10.4 Cross Sectional Shape

This group of parameters controls the shape of the inner channel's cross section on the Y-Z plane.

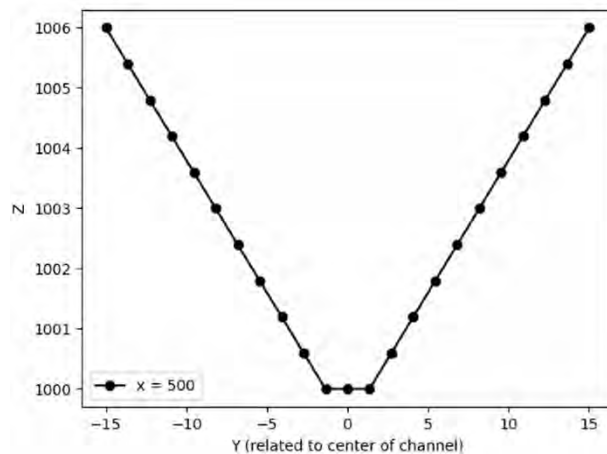
- **Channel XS Points** – This parameter controls two things. First, it controls the number of equally spaced points defining the inner channel's cross section for all underlying computations. However, this is not carried forward to the final topographic point output file. Instead, these results are interpolated to the inner channel grid dictated by the X Resolution parameter. Second, it controls the number of equally spaced points defining the inner channel's cross section for making the graph shown in the file, "SRVinnerChannelXShape.png". Input must be a positive integer greater than 1. For field data collection in a U-shaped channel, people commonly use ~ 15-30 points. An odd number is wise if you want to have a line of output points exactly in the center of the channel. The default value is 21.
- **Cross-Sectional Shape** – This is where the user decides whether to have an asymmetrical (AU), a symmetrical U shape (SU), or a symmetrical trapezoid (EN) cross-section.
  - o AU sets an asymmetrical cross-section. Once this choice is made, there is no further control over how the asymmetry works. That is dictated by the equations in River Builder. The methodology is explained in the third bullet point of the "Major Fixes" list (section 5.2) and the equations are provided in section 16.3.
  - o SU sets a symmetrical U shape cross-section.
  - o EN sets a symmetrical trapezoid cross-section, and then the user further controls the shape of that using the TZ(n) input next. These options are triangular, trapezoidal, or rectangular.
- **TZ(n)** – number of break points placed on the riverbed at the maximum depth across the inner channel's cross-section, not counting the left and right bank tops. For reference, see the far right cross-sectional diagram in Figure 16. This is only required when one uses SU. If these values are present when Cross-Sectional Shape is set to AU, then these values are

ignore and have no effect. In this function,  $n$  is the number of edges that defines the length of the trapezoidal base such that  $0 \leq n \leq (Y-Z \text{ Increments})$ .

- When  $TZ(n)=1$ , there is only one breakpoint at the centerline, so that yields a triangular cross-sectional shape
- When  $TZ(n) = \text{Channel XS Points}$ , then all of the breakpoints are along the bed right up to the banks, so that means the shape of the cross-section is rectangular
- When  $TZ(n) =$  any other number between 1 and Channel XS Points, then the shape is going to be trapezoidal. The higher the number, the steeper the banks.
- If you want to control the width of the flat bed of the trapezoidal channel, then you have to adjust the number of  $TZ(n)$  points carefully. Remember that the overall channel width is from bank top to bank top, so that width is always larger than the width of the channel bed between the sloped banks.
  - The formula for bottom width is:
  - $TZ(n) * [\text{Inner Channel Lateral Offset Minimum}] / [\text{Channel XS Points}]$ .
  - Therefore if you want to specify the bottom width to an exact number, you have to set
  - $TZ(n) = [\text{Bottom width}] * [\text{Channel XS Points}] / [\text{Inner Channel Lateral Offset Minimum}]$
- As a result, because  $TZ(n)$  has to be a whole number, then the best way to control bottom width is to set Channel XS Points = Inner Channel Lateral Offset Minimum and have both of those be an odd number.

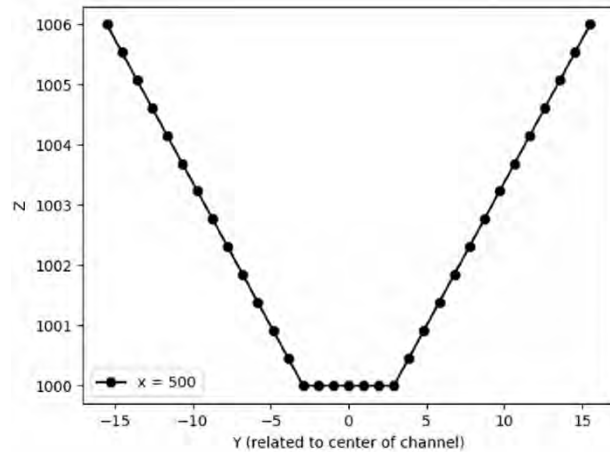
#### CASE 1

Inner Channel Lateral Offset Minimum = 30  
Channel XS Points = 21  
 $TZ(n) = 2$



#### CASE 2

Inner Channel Lateral Offset Minimum = 31  
Channel XS Points = 21  
 $TZ(n) = 6$



**Figure 21. Examples of trapezoidal cross-sections for different input parameters to obtain different bottom widths.**

## 10.5 User-Defined Functions

This group of parameters consists of the mathematical functions a user may then apply to any

feature (i.e., breakline) later on. It is common that a user might want to re-use the same function to control multiple features, so that is why these are specified independently from the river features.

Take note that if you do NOT want to have any sub-reach variability, then you can leave all of these inputs blank and the code will move forward using only straight lines for every breakline you set up.

By far, this is the most challenging part of river design, because River Builder does not tell you what functions with what parameter values to use- you have to decide that before getting started on the basis of your expertise as a river scientists and/or engineer. Further, it is especially difficult, because as of today very little of river science has looked into the structured patterning of sub-reach variability functions, except for riverbed undulation. Thanks to the common practice of mapping thalweg profiles, riverbed undulations are relatively well described in the literature. However, how those undulations relate to other sub-reach variability functions (i.e., their GCSs) has hardly been investigated yet. Further, there is relatively little about width variability and almost nothing about sub-reach variability functions for other breaklines outside the inner channel. If anything, this software serves as a call for more of such research to be done. However, as a user, you have to make do with what you can achieve right now and try your best to specify the sub-reach variability functions you think need to be there or which match real observations from reference reaches.

For each type of function, you first specify a text identifier for that type of function and then you specify a sequential numbering after that for each additional function of that type that you add. For example, the first sine function (SIN1) could have an amplitude of 2 and the second sine function (SIN2) could have an amplitude of 4. There is no limit to how many of each type you have, but each of the same type should have a different number. We recommend that users number functions upward sequentially, but there is no limitation to how you order and/or group numbering as long as each function has its own number. There is also no limit to how many different types you create, up to all the types that exist in the software at this time.

Each function has some number of parameters to make it work and the user must specify the values for all required parameters for a given function.

#### 10.5.1 Function text file option

River Builder is designed to have all inputs provided in a single text file, but this can get difficult as the sub-reach variability function for an individual breakline may be composed of as many additive components as you wish. As we have gotten experienced with reverse engineering complex breakline functions from real river data, we have found that it requires 20-200 frequency components to reconstruct the complex structure of key breaklines in natural rivers (Figure 14). What we have learned to do is use an analysis tool to extract the functions from real data and write them into a text file that is the output of that analysis tool. From there, we would copy/paste the text into the River Builder input file. That way works.

As an additional approach, River Builder 1.2. provides an easier way for users to input a large number of functions, especially as output of a reverse engineering algorithm. User can store functions for a breakline in a separate txt file, with each function on one line. Then, put this text file in the folder where you will run the River Builder command. Finally, you type the name of this file as a value in the River Builder input file. Note: in this way, you do not need to worry about the function sequential numbering conflicting with functions elsewhere, the program will handle it

automatically. Also, you do not need to specify the functions in the User-Defined Functions section. The follow is an example for this use.

In a text file called **cl\_functions.txt**, we have:

```
SIN0=(5.2, 1, 3.1, MASK0)
SIN1=(2.3, 2, -1.8, MASK0)
SIN2=(0.6, 3, -3, MASK0)
SIN3=(0.5, 4, -2.9, MASK0)
SIN4=(5.2, 1, 3.1, MASK0)
```

And in the River Builder input file, we have:

```
# User-Defined Functions section:
MASK0=(ALL)
SIN1=(1, 1, 0, MASK0)
```

```
Meandering Centerline Function=cl_functions.txt
Meandering Centerline Function=SIN1
```

In this example, user is using functions specified in cl\_functions.txt file and SIN1 function in User Defined Functions section together to create a centerline. Although there is a SIN1 in cl\_funtions.txt, it won't interfere with user defined SIN1, both of them will be counted. Nevertheless, it is best if you can avoid multiple uses of the same function ID in case you need to troubleshoot problems later.

Overall, these are the two ways you can import complex sub-reach variability functions into River Builder and rely on the software to protect you from function numbering conflicts.

### 10.5.2 Types of functions

At present there are 9 different kinds of functions available for you to use to create organized sub-reach variability.

#### 10.5.2.1 Periodic functions

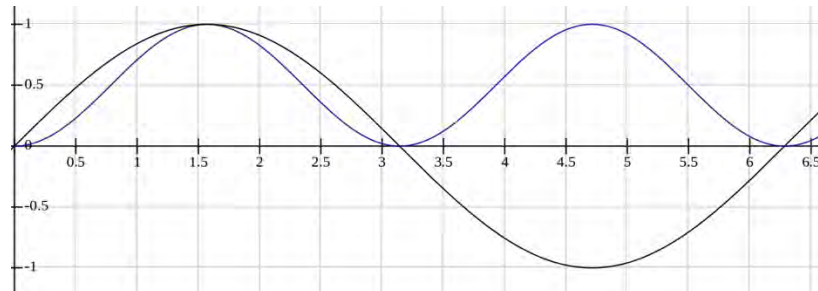
Seven types of functions (Sine, Cosine, Sine<sup>2</sup>, Cosine<sup>2</sup>, Cnoidal, Rectangular Step Wave, and Highly Curved) provide periodic oscillations. As simple as those are individually, remember that you can add as many of them as you want together into one overall function using addition. Subtraction can be easily achieved by using a negative amplitude. As shown in Brown et al. (2014), it is possible to get some very interesting and meaningful variability patterns with 2-4 SIN() functions added together, using different values for frequency, amplitude, and phase. One way to ascertain what those parameter value could be would be to perform harmonic analysis on real series of river variables, such as bed elevation, width, meander centerline, etc. You can extract all the periodic functions you want from that kind of analysis and use those in your design to whatever degree you wish.

- The sine and cosine functions are widely known and have been used to conceptualize river designs for generations. It is important to understand that while it is easy to conceptualize a river with these functions, few rivers actually vary exactly according to their simple geometry.



By using multiple frequencies of these functions, you can get a better representation of reality, but River Builder does not limit you to that. In any case, these functions are a good starting place to consider.

- The  $\sin^2$  and  $\cos^2$  functions are valuable because they produce two positive peaks over the range for which the sine and cosine functions only produce one positive peak. That could be used to enforce that something repeats for each location of max/min position along a sine wave, not only at the maximum or only at the minimum.



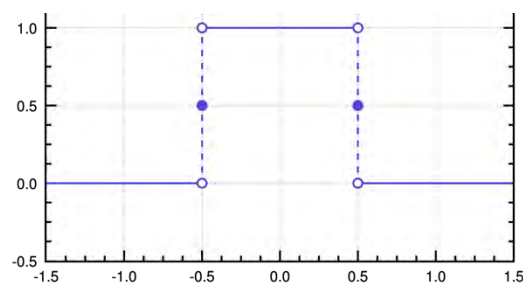
**Figure 22.  $\sin^2$  function (blue) compared with sine function.**

- The Cnoidal function allows for creating a periodic function with longer, flatter zones separated by more peaked zones as compared to the sinusoid function. Parameters control how long and flat the flat zones are. When taken to the extreme and applied to the inner channel, the Cnoidal function can be used to make sharp-crested weirs for the longitudinal bed profile and/or plan view jetties.



**Figure 23. Cnoidal function example.**

- The nearly rectangular step wave function allows for creation of flat “treads” separated by nearly straight “risers” on the basis of a representation using many cosine functions. In a thalweg profile this could be used to have a series of bed sills or broad-crested weirs (though we now have a separate dams tool for that need). In a plan view breakline it could be used to make a bedrock outcrop or man-made bank feature, such an artificial peninsula or jetty.



**Figure 24. Ideal rectangular step . The step wave function approximates this but maintains a**

unique y or z value for each x coordinate.

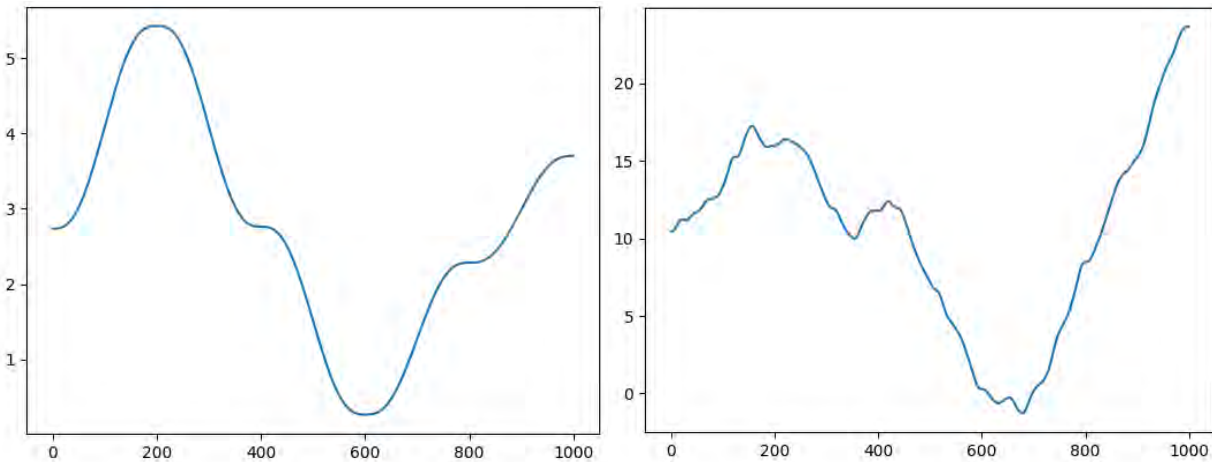
- The Highly Curved wave is the function you use when you want meanders with multiple y values for each x coordinate, which are otherwise known as gooseneck or looped meanders (Figure 17).

#### 10.5.2.2 Line function

Because rivers often do exhibit linear trends in variables, such as width and depth, River Builder enables you to apply the line function for use to create that effect.

#### 10.5.2.3 Non-periodic, deterministic oscillations

Sometimes you want to have a function that is deterministic and oscillating, but non-periodic. One application for this type of function would be for the meander centerline, because channel alignment is not going to be truly periodic. To achieve this, River Builder includes the three-parameter Perlin noise function. To find out what sinuosity you get from a particular set of Perlin parameters, you have to run River Builder and look at the SRVmetrics.txt file. That's a bit clunky, but it works. Alternately, you can use an online Perlin noise function calculator to fine-tune what you want and then simply copy the parameters from that calculator into your River Builder input file.



**Figure 25. Perlin function examples. Both use the same amplitude and wavelength, but the one on the left only has 1 octave while the one on the right has 4 octaves. More octaves means the function has more rugosity.**

#### 10.5.2.4 Piecewise functionality

River Builder is generally intended to serve as a reach-scale river valley design tool. As a result, if you want to create a longer river segment composed of multiple reaches, then it makes the most sense to create a unique design for each reach. However there are a few time when breaking a long channel into multiple independent files just does not make sense. The easiest example of this for a

natural river is a step-pool channel type. Nobody would want to make one reach design for each section between drops and then one reach design for the riser of each step. What if you wanted to make a river design of Niagara Falls. The second example of where this comes into play involves practical river design wherein there exist non-adjustable fixed points that serve as lateral turning points for the centerline or as vertical change points, like when one might want to daylight a stream at a certain elevation. After thoroughly studying the math of continuous functions with discontinuities, we decided that the best solution to the problem of discontinuities was to add a piecewise functionality to River Builder (Figure 19).

The way this work is that every geometric variability function has a fourth parameter called “MASK”. MASK is an entire function unto itself with its own novel parameterization. Just like with the variability functions, you create a numbered MASK function in sequential order, MASK1, MASK2, MASK3, etc. By default, every input file must include MASK0, which states that the geometric variability function applies to the whole length of the river. Consequently, by default every function can begin with MASK0 as its fourth parameter.

MASK functions include just two parameters, but they can repeat many times in sequence in the function. The first parameter is a position along the centerline. The second parameter states whether the MASK is on or off beginning at the position in the first parameter. After that, the user writes each change position and whether the mask is turning on or off for the length of the river.

For example, imagine a 1-km reach in which the first 250-m section was engineered straight and then after that it meandered. There is no need to specify on/off at position 0. You would create a line function with a mask that turns it off at position 250. The nomenclature for this would be “MASK#=(250,off)”, where # is the MASK function number you assign to this. You would also create a sine function with a mask that is off at 0 and then turns on at 250. The nomenclature for this would be “MASK#=(250,on)”, where # is the MASK function number you assign to this. For a single change/turning point, it is that simple.

Note that each section of a river can have as many geometric variability functions as you want to specify for it, and you can simplify the process by re-using the same MASK function for all the geometric variability functions that will be turned on and off at the same positions.

The problem with this nomenclature is that if you want to make a complex river, then it can get quite involved to specify all the mask functions. It is possible, but time-consuming.

For now, this is how River Builder works.

A future feature that is under development is a set of “function generators” that have a much-simpler way for you to specify typical types of piecewise channels. For now, there is only a step function generator to create sequences of step-pool units very easily. This tool is a separate Python3 script that takes your step-design inputs and produces the exact geometric variability and MASK functions you need for River Builder to get what you want. There are two ways you can use the output of this tool. First, you can copy/paste the output text into your River Builder input file. Second, you can type the tool’s output file name where designated in the River Builder input file to have it read it (see section 10.5.1).

### 10.5.3 Function parameters

Many functions tend to use the same types of parameters, even if they mean different things for different functions. All parameter values are in absolute units, but not referenced to any specific unit system, so it does not matter if you are working in SI, American customary units, or anything else. How a function parameter works with its absolute units depends on which channel breakline it is being applied to, and that is explained in detail in the list of parameters next. Special for periodic functions, the whole reach is treated as one period cycle, which is  $2\pi$ . This even holds for  $\sin^2$  and  $\cos^2$  functions, whose period is  $\pi$ . Different period cycles can be obtained by manipulating the frequency parameter. If this becomes confusing, the solution to learn how it works for yourself is to play with a simple design using a single geometric variability function applied to one channel breakline in an input file to see how it works.

Here is a list of the parameters

- **Amplitude (a)** – The range that the function’s magnitude will span in absolute units. The amplitude is defined in an absolute scale for the variable the function is being used for. How it works will depend on the native range of the function it is being used in. For example, if used for thalweg elevation, then a value of 1 changes the elevation by 1, regardless of the function. However, as another example, if you want to have a 50-m undulation using specifically the sine function, then you would set the amplitude to 25, remembering that the sine function goes from -1 to 1. If it was the  $\sin^2$  function, then that only goes from 0 to 1, so a with undulation of 50 would require an amplitude of 50. The amplitude applies to all functions that call for it as a parameter, but you have to be mindful of the native range of the function you are using.
- **Frequency (f)** – How many times the function will repeat itself over a length scale. In this software, the length scale depends on what this function is being used for. A design begins with a valley length and centerline, so the scale of that centerline is the length of the valley. After that, if the centerline has a variability function, then anything dependent on that function will be referenced to the length of the centerline. For example, undulations in inner channel width will go along the meandering centerline scaled by the length of the meandering centerline. Inner and outer bank breaklines will be scaled by the channel centerline’s length, while valley level breaklines will be scaled by the valley’s centerline. No matter the length of the centerline a breakline’s undulations is based counting on, the number of periods will always equal the number of frequency
- **Phase shift (ps)** – The position along the function where the function will begin. This is referenced relative to the start of the periodic cycle.
- **Wavelength (w)** – Some functions use wavelength instead of frequency. This is defined as the distance over which the function’s shape repeats.
- **Octave** – The number of times that the Perlin function is iterated at smaller spatial scales. The higher this number, the greater the rugosity a function will have (Figure 25).
- **Slope (slope)** – the rise or fall per unit length. Positive value is oriented such that the river

flows downstream and downslope. Positive stands for increasing the distance along meandering stream from upstream to downstream.

- **Y-intercept (y-intercept)** – The offset of a linear function when X equals zero.
- **Elliptic parameter (m)** – used in Cnoidal and Step functions. m should have a value between 0 to 1. When m = 0, it becomes cosine function; when m ~ 1, it has sharp peaks or steps.
- **Sinuosity parameter (p)** – used in high curvature function. It should be a positive number. Typically, with p > 2, functions show in high curvature feature.
- **Mask position (DIST)** – position along the centerline governing the breakline of interest where a mask is turned on or off.
- **Mask on/off** – a text identifier indicating whether a mask is turning “on” or “off” at the specified centerline position.

#### 10.5.4 Available Variability Functions

For each function listed below, a brief definition is provided. Indented bullets show the exact notation to be used in the user-defined functions section of the input file. The parameter symbols match those in the previous section.

- **MASK** function. A special kind of function, MASK function, is integrated with other functions as a parameter. A recognizable mask function should have the following format:

MASK#=(DIST1, on/off, DIST2, on/off, DIST3, on/off, ...)

- DIST means starting from what distance we want to turn on/turn off the function.
- DIST should always increases.
- DIST should not exceed the maximum distance of the river. (Exceeding parts will be ignored)
- DIST should always be integers.
- Every DIST should pair with an on/off switch.
- A default MASK function MUST always be defined: **MASK0=(ALL)**. It means the function will always be turned on for the whole length of the river.

An example mask with a 200-long reach could be:

MASK1=(0, off, 30, on, 75, off, 135, on) \*long-form version

MASK1=(30, on, 75, off, 135, on) \*short-form version

(both of these yield the exact same outcome, as by default River Builder understands that if you turn on a mask at a certain point then it must have been off before that)

- Repeated for clarity and emphasis: **MASK0=(ALL)** is always required in your input file.

(Mathematical equations for functions listed below can be found in the Appendix)



- **Sine** and **cosine** function. Standard trigonometry as you expect. Requires specification of amplitude, frequency, and phase
  - o  $\text{SIN\#}=(a, f, ps, \text{MASK})$
  - o  $\text{COS\#}=(a, f, ps, \text{MASK})$
- **Sine<sup>2</sup>** and **cosine<sup>2</sup>** function. Standard trigonometry as you expect. Requires specification of amplitude, frequency, and phase
  - o  $\text{SINSQ\#}=(a, f, ps, \text{MASK})$
  - o  $\text{COSSQ\#}=(a, f, ps, \text{MASK})$
- **Linear** function. Standard line. Note that the riverbed already has a built in slope function, so you do not have to add this to the thalweg to have a sloped riverbed. Could be used to create width expansions or constrictions, as one example.
  - o  $\text{LINE\#}=(\text{slope}, y\text{-intercept}, \text{MASK})$
- **Perlin** function – A smoothed, interpolated noise function that resembles a natural fractal curve. The function first generates a series of random numbers and then fits a curve to the set. The amplitude parameter controls the magnitude range of the random points, while the wavelength parameter controls the spacing between points. Next, the process is repeated for incrementally changed values of amplitude and frequency to obtain a set of curves. Finally, the curves are summed to yield a single function with many scales of fluctuations, similar to a fractal function. The persistence parameter controls how amplitude and wavelength change for a given set of curves to obtain different patterns of complex curvature. This function is especially suitable for meandering the centerline to avoid a pure sinusoidal shape.
  - o  $\text{PERLIN\#}=(a, w, \text{octave}, \text{MASK})$
- **Cnoidal** function – Periodic wave with one side flatter and one side sharper. If this was used for the riverbed, then the pools would be longer and flatter, while the riffles would be shorter and more peaked- or vice versa. See. Figure 19. If  $m=0$  it will be a cosine function.
  - o  $\text{CNOIDAL\#}=(a, f, ps, m, \text{MASK})$
- **Rectangular** step wave function – Allows for creation of flat “treads” separated by perfectly straight “risers”. If  $m=0$  it will be a cosine function.
  - o  $\text{STEP\#}=(a, f, ps, m, \text{MASK})$
- **Looped** “Highly Curved” wave function – Gooseneck meandering.
  - o  $\text{HIGHCURV\#}=(a, f, ps, p, \text{MASK})$

#### 10.5.5 Example List of Functions

Once you select which functions you want to use and specify the parameter values for each instance of each function, then the next step is to write out your list of functions.

The list goes in the section of the input file named "user-defined functions".

A list of functions might be something like this:

MASK0=(ALL)  
MASK1=(0, off, 500, on)

SIN1=(1, 4, 0, MASK0)  
SIN2=(0.25, 5, 0, MASK0)  
COS1=(0.5, 4, 0, MASK0)  
COS2=(0.5, 2, 0, MASK0)

SIN3=(0.3, 4, 0, MASK0)  
SIN4=(0.1, 2, 0, MASK0)  
COS3=(0.2, 4, 0, MASK0)  
COS4=(0.1, 2, 0, MASK0)

SIN5=(175, 0.5, 0, MASK0)  
SIN6=(50, 2, 0, MASK0)  
COS5=(75, 4, pi, MASK0)  
COS6=(25, 10, 0, MASK0)

SIN7=(100, 0.2, 0, MASK0)  
SIN8=(50, 1, 0, MASK0)  
COS7=(75, 2, pi, MASK0)  
COS8=(10, 7, 0, MASK0)

SIN9=(100, 0.2, 0, MASK0)  
SIN10=(50, 1, pi, MASK0)  
COS9=(75, 2, 0, MASK0)  
COS10=(10, 7, 0, MASK0)

SIN11=(150, 1, 0, MASK0)

Notice that in this example list (taken from a variant of our “S10” archetype) there are four sets of pairs of sine and cosine functions used, plus an additional sine function at the end. This is done to organize subsets that will go together for sets of river breaklines that will share the same sub-reach variability functions. It is not required to do it this way. You can list them in any order you want. Whatever works for you.

## 10.6 Channel Breakline Input Parameters

This is where you get down to business with specifying sub-reach variability functions and their specific parameter values for each channel breakline in your design.

Take note that if you want a design with no sub-reach variability and no breaklines outside the inner channel, then you can leave this whole section blank and you will get a straight vanilla channel, such as shown in Figure 1b.

**Note:** Do not modify any of the names of the parameters in the input text files (doing so will result in failure to use the functions or values correctly).

### 10.6.1 Channel breaklines

This section defines the geomorphic aspects of the river that variability functions may be used to control. The channel features are broken into two groups, those that describe the inner channel and those that describe the outer channel.

#### 10.6.1.1 Inner channel properties

There are five optional river features that control sub-reach variability of the inner channel in plan view. Each of these features can have a sub-reach variability function.

- **Meandering Centerline** – governs the shape of the river’s tortuous flow path (i.e. meandering or sinuosity) on the X-Y plane. Note that the thalweg does not necessarily occur on the centerline.
  - o Meandering Centerline Function=
- **Centerline Smoothness** – this is an optional feature that governs the smoothness of the centerline. This feature prevents sharp turns in the centerline, specifically when a piecewise mask function is used, as that tends to have such sharp breaks. There is nothing wrong with sharp breaks per se, but when you make a raster DEM from a point set for such a design, it tends to look bad at the sharp breaks. It takes a positive integer value as input where the value is the number of adjacent points to be included in a moving average window that smooths the centerline coordinates. For example, a value of 1 means taking the average of a point at its nearest adjacent points to the left and right (so it will be 3 points total when the input is 1). The higher the value, the more smoothing occurs. Smooth=0 means no smooth at all. Whenever a piece-wise mask function is applied to the centerline, at least Smooth=1 is recommended.
  - o Smooth=
- **Centerline Curvature** – Normally the sub-reach variability function for centerline curvature would be set equal to the computed curvature function governing the meandering centerline. However, one application of the curvature value is with controlling the asymmetry of a channel’s cross-sectional shape. To provide greater flexibility with how a river’s XS shape changes along the river, we have created a decoupled centerline curvature variable as an option. Normally one would not use this, but for an advanced user it might be considered. This variable has no effect on the channel’s meandering. It only affects how channel cross-sectional asymmetry varies down the river along the meandering. For example, let’s say that you did not want the deepest part of the cross-section to be at the outer bend of a meander, then you could use this to put the deepest part anywhere you want governed by the available variability functions. That is what this is for. This is a highly complex tool to control compared to others, so only use it if you really know what you are doing with your river design concept.
  - o Centerline Curvature=
- **Thalweg Elevation** – governs downstream undulations in the bed elevation along the thalweg on the X-Z plane. May be used to create simple riffle-pool couplets and/or more complex sets of morphological units.

- Thalweg Elevation Function=
- **Inner Bank** – The next river feature is actually a pair of breaklines used to specify inner channel width variability- **left inner bank** and **right inner bank**. Left Inner Bank governs the downstream variability in the river-left bank top breakline on the X-Y plane. There is only one left inner bank. Right Inner Bank governs the downstream variability in the river-right bank top breakline on the X-Y plane. There is only one right inner bank. To provide maximum flexibility, the user specifies each bank independently with as many user-defined functions as desired, but of course the two banks can be assigned the same function(s) if you want them to behave the same way and have a symmetrical outcome. Together, the left and right inner bank functions define inner channel width. Recall that the “Inner Channel Width Offset Minimum” parameter will keep the two inner bank functions from crossing over as long as it is not set to zero. Note that the two inner bank inputs both go under this one heading, so you may want to separate them with an empty line but that is optional.
  - Left Inner Bank Function=
  - Right Inner Bank Function=

Ideally, one might expect that for maximum freedom between the two banks, the user ought to be able to not only specify different plan view sub-reach variability functions, but also different bank top elevations. The software has the capability with a workaround, but not using the inner channel specifications. The reason why it should not be attempted in coding the inner channel, is because we also must have the river’s cross-section specified by a function and it would get extremely more complex if that XS shape function had to be cut in half to yield different bank tops for the left and right sides while still meeting exactly together at the centerline. That’s too much unnecessary coding to figure out compared to the solution we have implemented instead.

The simpler solution is to lock the inner channel left and right bank top elevations to a single value (governed by the specified the Inner Channel Depth Minimum parameter) and then let the user specify additional “outer bank” breaklines, each with their own vertical and lateral offsets to create whatever asymmetric banks is desired. This will become cleared after explaining the next two inner channel properties.

#### 10.6.1.2 Outer channel properties

The outer channel is the zone between the bank top and the inner edge of the valley zone (Figure 7). There are two of these zones- left and right. “Outer banks” are individual breaklines that are within an outer channel zone. They may be used to represent asymmetric bank heights for the inner channel (by simply creating another bank breakline close to the inner bank but at whatever additional positive height offset is desired) and/or represent additional lateral slope breaks and channel features for a multi-level nested channel system. In dry climates it is common to have lower flow channels carved inside higher flow channels. Australian “macro-channels” with multiple nested channels within channels are the extreme of this in reality.

River Builder allows you to have as many outer bank breaklines as you wish, and there are many novel ways to apply them to get different outcomes. The simplest examples is to have nested channels within channels. However, mindful use of multiple positive and negative height offsets combined with high-amplitude sub-reach variability functions could yield braided, anastomosing, an

yazoo channel scenarios. There can only be one main channel with thalweg undulation at this time, but conceptually having multiple threads or complex braided threads is plausible, though we have not attempted to create these archetypes for ourselves as of yet.

Specifying outer banks work a bit differently from the inner banks in two ways. First, there is an infinite number permitted, so it is necessary to have a numbering system. This handled by using a one-letter designator (L or R) for which side of the river one is addressing followed by a sequential number. For example, for 3 breaklines on the left side one would use the designators L1, L2, and L3. The designators for left and ride sizes are independent and do not have to match in height offset, so L3 and R3 can be at totally different elevations (or there can be an R3 with no L3 at all).

Second, the lateral and vertical positioning of these breaklines must be specified, and this is done relative to the centerline. For the inner banks, lateral and vertical positioning were handled by the width and depth values specified in the Channel Parameters section. For each outer bank breakline, you have to specify its offsets from the centerline (Figure 8b). An offset is the distance from the centerline to the breakline. Since it refers to the centerline and each breakline may have a unique sub-reach variability function, there is no easy way to figure out whether breaklines overlap with each other in River Builder v 1.2. Thus, you need to carefully choose your parameters to avoid that and then check the 2D plan-view output plot to confirm that there are no overlaps.

The notation for the lateral offset is “Outer Bank Offset Minimum”. For example, “L1 Outer Bank Offset Minimum=2”. The lateral offset has to be a minimum, because one can apply a sub-reach variability function to it. The notation for the height offset is “Outer Bank Height”. For example, “L1 Outer Bank Height=10”. There are no vertical sub-reach variability functions in River Builder for anything other than the centerline.

Finally, outer bank breaklines can be assigned sub-reach variability functions using the same approach as for inner channel features, as previously discussed.

- **Left Outer Bank** – Breakline in the left outer channel zone.
  - o L1 Outer Bank Function=
  - o L1 Outer Bank Lateral Offset Minimum=
  - o L1 Outer Bank Height Offset=
  - o Increment as L1, L2, L3, etc. as far as you wish to go, every numbering must be used only once and must be in increasing order with distance away from channel center.
  - o After the equal sign for the two offset functions you write the offset value in units of meters. For the variability function, see section 10.6.2
- **Right Outer Bank** – Breakline in the right outer channel zone.
  - o R1 Outer Bank Function=
  - o R1 Outer Bank Lateral Offset Minimum=
  - o R1 Outer Bank Height Offset=
  - o Increment as R1, R2, R3, etc. as far as you wish to go, every numbering must be used only once and must be in increasing order with distance away from channel center.
  - o After the equal sign for the two offset functions you write the offset value in units of meters. For the variability function, see section 10.6.2



### 10.6.2 Assigning Functions To Channel Sub-Reach Variability Parameters

The way to assign a function to a sub-reach variability parameter is to simply write the name of the function right after the equals sign on each line. For example,

Right Inner Bank Function=SIN3

If you want to add multiple user-defined functions together, simply list all the functions on subsequent lines. For example,

Right Inner Bank Function=SIN3  
Right Inner Bank Function=SIN4  
Right Inner Bank Function=COS3  
Right Inner Bank Function=COS4

As another example, here is a list with offsets and variability functions:

L1 Outer Bank Lateral Offset Minimum=5  
L1 Outer Bank Height Offset=15  
L1 Outer Bank Function=SIN7  
L1 Outer Bank Function=SIN8  
L1 Outer Bank Function=COS7  
L1 Outer Bank Function=COS8

## 10.7 Valley Breakline Input Parameters

One of the cool features of River Builder is the capability to nest the channel within a valley that has all of its own independent features. Figure 4 illustrates a real river nested in a sinuous valley. River Builder can replicate that by assigning a sub-reach variability function to a valley centerline that is independent of the channel centerline. In addition, the valley can have as many plan view breaklines as you want to implement to create features such as floodplain levels, terraces, roads, levees, yazoo channels, etc.

The way the valley zone works is very similar to how the outer bank zone works:

- The resolution of valley accords with that of channel.
- There is a valley centerline to serve as a baseline in calculation of elevation of valley breaklines.
- There are independent left and right valley zones.
- Each zone has a L and R designator.
- Each breakline in a valley zone gets a sequential number increasing with distance away from the channel.
- Each breakline has three attributes: a lateral offset, a height offset, and (optionally) a sub-reach variability function.

### 10.7.1 Valley Breaklines

- **Valley Centerline**– governs the shape of the river’s tortuous flow path (i.e. meandering or sinuosity) on the X-Y plane. The channel’s centerline is nested within the valley’s streamwise coordinate system defined by this function.
  - Valley Centerline Function=
- **Left Valley Level** – Breakline in the river-left valley zone. Offsets are referenced to the valley centerline.
  - L1 Valley Level Lateral Offset Minimum=
  - L1 Valley Level Height Offset=
  - L1 Valley Level Function=
  - Increment as L1, L2, L3, etc. as far as you wish to go. Numbering must be in increasing order with distance away from valley center.
  - After the equal sign for the two offset functions you write the offset value in units of meters. For the variability function, see section 10.6.2.
- **Right Valley Level** – Breakline in the river-right valley zone. Offsets are referenced to the valley centerline.
  - R1 Valley Level Lateral Offset Minimum=
  - R1 Valley Level Height Offset=
  - R1 Valley Level Function=
  - Increment as R1, R2, R3, etc. as far as you wish to go. Numbering must be in increasing order with distance away from valley center.
  - After the equal sign for the two offset functions you write the offset value in units of meters. For the variability function, see section 10.6.2.

The next two sets of valley breaklines are different, because they define the hard outer boundary of the domain. These “Valley Boundary” breaklines are always parallel to the valley centerline and thus no functions will apply to them. Only offsets need be specified, and these offsets are NOT referenced to the centerline, but to the adjacent breakline. If the user forgoes specifying boundary offset values, the program automatically assigns default values of 10 for the lateral offset and 20 for the height offset.

- **Left Valley Boundary** – The final, outermost breakline in the river-left valley zone. There is nothing beyond this one, so it is the outer boundary of the terrain model. Because it serves mostly to provide a friendly boundary for 3-D rendering software like GIS, it is designed that user cannot set functions to it.
  - Left Valley Boundary Lateral Offset Minimum=
  - Left Valley Boundary Height Offset=
- **Right Valley Boundary** – The final, outermost breakline in the river-right valley zone. There is nothing beyond this one, so it is the outer boundary of the terrain model. Because it serves mostly to provide a friendly boundary for 3-D rendering software like GIS, it is designed that user cannot set functions to it.
  - Right Valley Boundary Lateral Offset Minimum=
  - Right Valley Boundary Height Offset=

### 10.7.2 Assigning Functions To Valley Sub-Reach Variability Parameters

The procedure for assigning sub-reach variability functions for the valley breaklines is identical to that used for the channel, so refer to section 10.6.2.

Here is an example:

L1 Valley Level Lateral Offset Minimum=5  
L1 Valley Level Height Offset=15  
L1 Valley Level Function=SIN7  
L1 Valley Level Function=SIN8

## 10.8 Optimization Routine

All parameters require starting values to run the program. Conceptually you might expect that these are the reach-average values, but of course one you apply any one or set of geometric variability functions at channel and valley scales, then it becomes extremely difficult to state in advance what the reach-average values are in the end. If you had to know that at the outset, you would need such a complex calculator that it would defeat the purpose of this software.

The goal of River Builder is to make your job easier and more complete, not much worse. To help you arrive at the final reach-average metrics you want, River Builder include an optimization routine that allows you to identify some reach average values as the final goal. Once these reach average values are defined, the program will automatically iterate with reasonable adjustments on some parameters until a halting condition is met. Of course, if you try to over-specify requirements, then it might become impossible to solve. The simpler the channel, the more you can confidently control and optimize for. If you make the variability functions highly complex, do not expect everything else to be easily forced to some set values.

There are 3 halting conditions to the optimization routine:

1. The calculated reach average value meets the required reach average value.
2. Number of iterations exceeds 100.
3. Unable to satisfy the reach average value requirements even with extreme parameters. For example, if calculated reach average value is greater than required reach average value when lateral offset is set to 0, program halts because lateral offset can't go to negative.

The resolution of reach average value is the same as user defined. For example, if a user defines river slope to be 0.1 (and this value is reasonable with given parameters), then the final calculated river will have a river slope **ROUNDED** to 0.1. (That is to say, from range 0.05 to 0.14.) If it is set to 0.10, then the final calculated river will have a river slope **ROUNDED** to 0.10 (range from 0.095 to 0.104).

Only inner channel parameters can have pre-defined reach average values, because outer banks and valley breaklines are not matched in pairs by definition. The following 3 reach average values can be pre-defined for optimization:

- Inner Channel Average Bankfull Width

- Inner Channel Average Bankfull Depth
- River Slope

Here is an example:

Inner Channel Average Bankfull Width=30  
 Inner Channel Average Bankfull Depth=10.5  
 River Slope=0.001

Optimization control commands are specified on lines at the end of the River Builder input text file.

Note: River slope highly depends on the sinuosity of the meandering centerline. Therefore user may ends with a river in different sinuosity than they expected with their inputs in User-Defined Functions section, if they want to iterate though this “River Slope” parameter.

## 10.9 Objects

The program provides some optional features that can enrich the inner channel bed. Currently they are bed elements, bed roughness, and dams.

### 10.9.1 Perlin Bed Roughness

As cross-sectional shapes are only calculated for the inner channel, bed roughness can only be applied to inner channel bed. Despite the shape of the inner channel bed, bed roughness will add a 2-D Perlin noise surface to it. The user should provide a positive number indicating the fluctuation. For example, PBR=5 means that the 2-D Perlin noise layer will have a height range from -5 to 5.

- PBR=

### 10.9.2 Bed Element Group

Bed elements can be protrusions or depressions, and they are organized as groups on the basis of size. Users can apply this feature to add bed elements as they wish. Bed elements are added RANDOMLY to the inner channel bed. The number of bed element groups is unlimited, as well as the number of bed elements in each group. However, bed elements from different groups can overlap with each other, but bed elements from the same group CANNOT overlap. Therefore, for each specific group of bed elements, there will be a maximum number of how many of them can be added due to the limited area of inner channel bed. For each bed element group, user can specify the number wanted to be added, the mean size from normal distribution, the standard deviation of the size from normal distribution, and the height (can be negative) of the bed elements.

- BEG1=(number1, size1, std1, height1)
- BEG2=(number2, size2, std2, height2)

### 10.9.3 Dams

Dams are specified individually using three parameters, x position along the channel centerline,

height above the riverbed, and thickness (i.e., crest distance from upstream to downstream). In the input file these are called “check dams”, but that does not mean anything. Depending on the height and thickness, they can be a variety of features.

- $CD1 = (\text{location1}, \text{height1}, \text{thickness1})$
- $CD2 = (\text{location2}, \text{height2}, \text{thickness2})$

## 11 Stepped Inner Channel Generator

This section of the manual explains how to use the independent tool, “functionGenerator.py”, which is part of the River Builder Github. This tool is used to greatly simplify the design of rivers with stepped inner channels. The general concepts were covered earlier in section 3.2, so this section will just discuss the set up and use of the tool.

The approach to using the tool is the same as for River Builder as a whole. The user begins with a template version of a text input file, enters the values for what they want to design, and then runs the tool to produce the output. In this case, the output is a text file called “steppoolThalwegFunctions.txt” that provides the user with lines of text that can be used in the main River Builder program to produce rivers with stepped inner channels. The program also outputs a longitudinal profile that illustrates what the stepped channel looks like.

The template of the text input file is called “functionGenerator\_format.txt”. This text file has pretty detailed instructions on how to use the tool.

For brevity, this section will assume knowledge from the previous sections and not repeat detailed instructions for steps that have already been explained.

### 11.1 Reach Parameters

The first action is to specify reach length and straight channel slope the same as used in the main River Builder input file. Then if the river is going to meander, then the functions to be used for the meandering centerline should be written out. This includes the use of any mask function.

### 11.2 Step Parameters

A step group is defined as a repeating tread-riser couplet. Treads and risers were previously defined. A step group is assigned values for tread length, riser height, riser length, and the shape of the tread function. The combination of riser height and length values constitute the linear slope of the riser; a riser cannot be exactly vertical. When it is very steep, it is a step. When it is set to a lower slope, then the better term is “bed-supported slide”. Bed-supported means that when water passes downstream of the crest, then it still rests on the bed without launching out over it to create an air gap underneath. With a steep riser, one would expect such an air gap and a free-forming “nappe” waterfalls. These concepts are detailed and illustrated in Pasternack et al. (2006).

A tread function is very important, because it can be used to control what the morphological unit is for the length between each riser. For example, some rivers have plane bed treads, while others have deep scour holes at the base of each step. You can design the tread function to have the desired shape to replicate these various plane bed, pool, etc, lengths between risers. There can be as



many tread-riser couplets as you want in a step group (as long as they fit within the specific reach length), so the last parameter to specify is the number of repeats.

Step parameters are assigned for each step group in order from upstream to downstream.

That is all there is to it. Once you have the output from running this tool, you can choose to either paste it into your main River Builder input file or call to this output file from your main River Builder input file as explained earlier.

## 12 Generating 3D Renderings of River Builder Outputs

After you run River Builder in Python3, then you are going to want to visualize the output and then likely go through an iterative process of design refinement.

By design, River Builder is intended as a creation program, not an analysis program. It does not have any 3D rendering capability within it.

The key file to import into rendering software is **SRVtopo.csv**.

We have created a very simple tool for quickly converting the table of x, y, z coordinates of topography (e.g., see the output file SRVtopo.csv) to a TIN surface and raster DEM. If you go to the top level of the River Builder Github, then you will find a script named “RB\_to\_terrain.py”. This tool can be run to produce a raster DEM. Note that it requires “arcpy” module, so you must have an ArcGIS license to run this script. We have tried hard to avoid requiring any paid licenses to use River Builder, and this is why this tool is a separate add-on. This script can be executed by typing in the following command from the Windows command prompt:

```
"C:\Program Files\ArcGIS\Pro\bin\Python\Scripts\propy" rb_to_terrain.py
```

Within the script, if you open it using a text editor, you can edit the parameters of the terrain that is built. First, “RB\_unit” is the unit of the terrain created in River Builder and “asc\_unit” is the desired unit of ascii terrain. For both of these, you may type “meter” or “foot”, depending on which unit system you want to use. Next, “cell\_size” is the numerical value for the raster pixel size of the ascii terrain in asc\_unit. An array, “execute” indicates whether to execute each of four available steps or just do a subset of them. One can only do the later steps if the earlier steps are done as well. The four steps are (1) Table to point, (2) Create a TIN surface, (3) TIN to Raster, and (4) Raster to asc. This script will create “case\_name\_xyz.shp” (point shape file), “case\_name\_TIN” (TIN surface), “case\_name.tif” (TIF Raster topography) and “case\_name.asc” (ASCII Raster topography). Be sure to delete obsolete files before running this script as it does not overwrite the existing files. As always when running an automated script, you should review the output to make sure it has done what you want. Because River Builder includes outer valley boundary contours, your terrain will be well bounded and you should not need to worry about TIN interpolation affects within that boundary. However, depending on the complexity of the channel, you may find that to get a good interpolation for a raster that you need to make the “X resolution” value of the design smaller.

There are so many 3D rendering programs available already, both free and commercial that it has not seemed worth the cost and effort to take on writing our own code from scratch. Meanwhile,

if we choose to implement a commercial code for this, then we limit the user base if they do not have a license for that code. Therefore, we leave it to the user to take the simple output file with the topographic points and use it to render the terrain in whatever 2D mapping and/or 3D rendering program they want.

### 13 Acknowledgements

The underlying research used to develop River Builder came from a variety of sources, including UC Davis, the USDA National Institute of Food and Agriculture [Hatch project number #CA-D-LAW-7034-H], the Ticho Foundation (unrestricted donation), US Army Corps of Engineers, and Yuba County Water Agency (Award #201016094). Dr. Rocko Brown and Dr. Greg Pasternack also contributed their own resources in developing different aspects of the underlying research. We thank postdoctoral scholars Dr. Colin Byrne, Dr. Sebastian Schwindt, and Dr. Anzy Lee for conversations about various aspects of this version of the software as we were developing it. Dr. Lee also contributed the text for running the S1 example and for running the `rb_run.py` and `rb_to_terrain.py` scripts. Other members of the Pasternack, Sandoval (CD), and Lane (USU) lab groups gave input from time to time during 2019-2021.

### 14 References

- Blum M. D., Tornqvist, T. E. 2000. Fluvial responses to climate and sea-level change: a review and look forward. *Sedimentology* 47: 2-48.
- Brown, R. A., Pasternack, G. B. 2014. Hydrologic and topographic variability modulate channel change in mountain rivers. *Journal of Hydrology* 510: 551-564.
- Brown, R. A., Pasternack, G. B. 2017. Bed and width oscillations form coherent patterns in a partially confined, regulated gravel–cobble-bedded river adjusting to anthropogenic disturbances, *Earth Surface Dynamics*, 5, 1-20, doi:10.5194/esurf-5-1-2017.
- Brown, R. A. Pasternack, G. B. 2019. How to build a digital river. *Earth-Science Reviews*. DOI: 10.1016/j.earscirev.2019.04.028.
- Brown, R. A., Pasternack, G. B., Lin, T. 2015. The topographic design of river channels for form-process linkages for river restoration. *Environmental Management*, 57 (4): 929-942. doi: 10.1007/s00267-015-0648-0
- Brown, R. A., Pasternack, G. B., Wallender, W. W. 2014. Synthetic river valleys: creating prescribed topography for form-process inquiry and river rehabilitation design. *Geomorphology* 214: 40-55. 10.1016/j.geomorph.2014.02.025.
- Caamaño, D., Goodwin, P., Buffington, J. M.; Liou, J. C. P.; Daley-Laursen, S. 2009. Unifying criterion for the velocity reversal hypothesis in gravel-bed rivers. *Journal of Hydraulic Engineering*. 135(1): 66-70.

- Eubanks, C. E. 2004. Riparian restoration. US Department of Agriculture, Forest Service, Technology & Development Program.
- Grill, G., B. Lehner, M. Thieme, B. Geenen, D. Tickner, F. Antonelli, S. Babu, P. Borrelli, L. Cheng, H. Crochetiere, H. Ehalt Macedo, R. Filgueiras, M. Goichot, J. Higgins, Z. Hogan, B. Lip, M. E. McClain, J. Meng, M. Mulligan, C. Nilsson, J. D. Olden, J. J. Opperman, P. Petry, C. Reidy Liermann, L. Sáenz, S. Salinas-Rodríguez, P. Schelle, R. J. P. Schmitt, J. Snider, F. Tan, K. Tockner, P. H. Valdujo, A. van Soesbergen, and C. Zarfl. 2019. 'Mapping the world's free-flowing rivers', *Nature*, 569: 215-21.
- Jackson, J. R., Pasternack, G. B., Wheaton, J. M. 2015. Virtual manipulation of topography to test potential pool-riffle maintenance mechanisms. *Geomorphology* 228: 617-627.  
<http://dx.doi.org/10.1016/j.geomorph.2014.10.016>.
- Lane, B. A., Pasternack, G. B., Sandoval-Solis, S. 2018. Integrated analysis of flow, form, and function for river management and design testing. *Ecohydrology*. DOI: 10.1002/eco.1969.
- Lane, B.A., Pasternack, G.B., Dahlke, H.E., Sandoval-Solis, S. 2017. The role of topographic variability in river channel classification. *Physical Progress in Geography*. 18  
doi:10.1177/0309133317718133.
- MacWilliams, M. L., Wheaton, J. M., Pasternack, G. B., Kitanidis, P. K., Street, R. L. 2006. The Flow Convergence-Routing Hypothesis for Pool-Riffle Maintenance in Alluvial Rivers. *Water Resources Research* 42, W10427, doi:10.1029/2005WR004391.
- Pasternack, G. B., Wang, C. L., and Merz, J. 2004. Application of a 2D hydrodynamic model to reach-scale spawning gravel replenishment on the lower Mokelumne River, California. *River Research and Applications* 20:2:205-225.
- Pasternack, G. B., Baig, D., Weber, M., Brown, R. 2018a. Hierarchically nested river landform sequences. Part 1: Theory. *Earth Surface Processes and Landforms*. DOI: 10.1002/esp.4411.
- Pasternack, G. B., Baig, D., Weber, M., Brown, R. 2018b. Hierarchically nested river landform sequences. Part 2: Bankfull channel morphodynamics governed by valley nesting structure. *Earth Surface Processes and Landforms*. DOI: 10.1002/esp.4410.
- Pasternack, G.B., Ellis, C. Leier, K.A., Valle, B.L., Marr, J.D. 2006. Convergent hydraulics at horseshoe steps in bedrock rivers. *Geomorphology* 82:126-145.
- Schwindt, S., Larrieu, K. G., Pasternack G. B., Rabone, G. 2020. River Architect. *SoftwareX* 11: 100438. DOI: 10.1016/j.softx.2020.100438.
- Thompson, C.J., Croke, J., Fryirs, K. and Grove, J.R., 2016. A channel evolution model for subtropical macrochannel systems. *Catena*, 139, pp.199-213.

Wheaton, J. M., Pasternack, G. B., and Merz, J. E. 2004. Spawning Habitat Rehabilitation - 2. Using hypothesis development and testing in design, Mokelumne River, California, U.S.A. International Journal of River Basin Management 2:1:21-37.

## 15 Change Logs

With each update to River Builder there is a list of changes that have been implemented to improve it. As this is the first of version 1.0, there are no changes to report.

## 16 Mathematical Equation Appendix

This appendix contains a complete list of the mathematical functions used in River Builder to try to have some transparency in plain written text. Obviously, the code is open source, so everybody can study the equations and their implementation in the code directly, but it helps to have a list like this as well.

### 16.1 Geometric Variability Function Equations

- Sin and cos functions

$$f(a) = \text{amplitude} * \sin(\text{frequency} * a + \text{shift})$$

Where,

$$a = 2\pi * \frac{x_i}{x_{max}}$$

- $\sin^2$  and  $\cos^2$  functions

$$f(a) = \text{amplitude} * \sin^2(\text{frequency} * a + \text{shift})$$

Where,

$$a = 2\pi * \frac{x_i}{x_{max}}$$

- Cnoidal function

Set:

$$m \in [0, 1]$$

$$a = 2\pi * \frac{x_i}{x_{max}} + \text{shift},$$

then:

$$k(m) = \frac{1}{\sqrt{1 - m * \sin(a)}}$$

$$y_i = \text{amplitude} * \left( \frac{\cos(a)}{2} * k(m) \right)^2$$

$$y_i = y_{max} - y_i$$

- Perlin noise function

Each octave will decrease wavelength and amplitude into half.  
Within each octave:

$$y_{\lambda*(i+1)} = y_{\lambda*i} + r$$

$\lambda$ : wave length

$r$ : random number between [-amplitude, +amplitude]

Within each wave:

$$\alpha = \frac{x_i - x_{left}}{x_{right} - x_{left}}$$

$$\alpha = 6 * \alpha^5 - 15 * \alpha^4 + 10 * \alpha^3$$

$$y_i = (1 - \alpha) * y_{left} + \alpha * y_{right}$$

left: point at the beginning of the current wave

right: point at the end of the current wave

- Step function

Set:

$$m \in [0, 1]$$

$$a = 2\pi * \frac{x_i}{x_{max}} + shift,$$

then:

$$k(m) = \frac{1}{\sqrt{1 - m * \sin(a)}}$$

$$y_i = amplitude * (\frac{\cos(a)}{2} * k(m))$$

- High curvature function

$p$  = sinuosity

$s_i$  - point along meandering line.

$$\omega = 2.2 * \sqrt{\frac{p-1}{p}}$$

$$\theta = \omega * \cos(s_i)$$

$$x_{i+1} = x_i + \cos(\theta)$$

$$y_{i+1} = y_i + \sin(\theta)$$

## 16.2 River Bank/Valley Level Related Function Equations

- Width offset function

Left side:

$$offset_{yi} = y_i + fun(x_i) + min_{offset} - min(fun(x))$$

Right side:

$$offset_{yi} = y_i - fun(x_i) - min_{offset} + min(fun(x))$$

- Depth offset function

$$z_{i+1} = z_i + offset$$



## 16.3 Centerline Related Function equations

- Centerline calculation

$$y = fun_1(x) + fun_2(x) + fun_3(x) + \dots$$

- Shields equation

$$\underline{H_{BF}} = \frac{(\gamma_s - \gamma_w) \underline{D_{50} \tau_c}}{\gamma_w \underline{S}}$$

- SN-to-XY coordinate system transformation

$$x = -\frac{\Delta y}{\sqrt{\Delta x^2 + \Delta y^2}}n + x_1$$
$$y = \frac{\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}n + y_1$$

- XY-to-SN coordinate system transformation

$$\text{Left: } n = \sqrt{\Delta x^2 + \Delta y^2}; \text{ right: } n = -\sqrt{\Delta x^2 + \Delta y^2}$$

- Dynamic Curvature

$$radians = arccos\left(\frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{|\mathbf{v}_1| * |\mathbf{v}_2|}\right)$$

- Asymmetric XS shape points calculation

$$B = \frac{1}{2} * \left(1 - \left|\frac{cur}{cur_{max}}\right|\right)$$

$$L = -\log_B 2$$

$$Y = \frac{\frac{w_{BF}(s_i)}{2} - n}{w_{BF}(s_i)}$$

$$z_n(s_i) = 4 * h_{BF}(s_i) * Y^L * (1 - Y^L) \text{ if } cur \leq 0$$

$$z_n(s_i) = 4 * h_{BF}(s_i) * (1 - Y)^L * (1 - (1 - Y)^L) \text{ if } cur > 0$$

## 17 Troubleshooting

Inevitably, River Builder software will require further debugging and improvements as users create interesting scenarios that break the code in unexpected ways. Please email us your problems and we'll do our best to fix them. Obviously, this software is free and we have no consistent funding for software development, so we can only do what we can in our free time.

### 17.1 I set a mask function with some ons and offs, but the result is all on. Why doesn't the mask function work?

A recognizable mask function should have the following format:  
MASK#=(DIST1, on/off, DIST2, on/off, DIST3, on/off, ...)

DIST means starting from what distance we want to turn on/turn off the function.  
DIST should always increase.  
DIST should not exceed the maximum distance of the river. (Exceeding parts will be ignored)  
DIST should always be integers.  
Every dist should pair with a on/off switch.  
An example mask with a 200-long reach could be:  
MASK1=(0, off, 30, on, 75, off, 135, on)

Check the log to check whether the target mask function appeared in “User defined functions”; if not, it probably means that the format of target mask function is not correct or not defined.  
Check the spell and number of target mask function.

If the mask function appeared in “User defined functions”, then check if it has the correct format as stated above.

## **17.2 Why is my final ‘Average height of Inner Channel’ a negative number?**

Please check if both ‘Valley Slope’ and ‘Inner Channel Depth Minimum’ are set to 0. When ‘Inner Channel Depth Minimum’ is set to 0, depth will be calculated based on Shield’s Equation, and Shield’s Equation has to have a positive slope input. Thus we recommend to manually set ‘Inner Channel Depth Minimum’ if user wants to set slope to 0.

## **17.3 I used some variability functions on channel centerline, but no matter how I tuned the amplitudes, I always get the same image.**

Please check if “River Slope” parameter in “User Requirements” section is set. “User Requirements” setting has the highest priority, so the program will iterate through centerline function amplitudes to try satisfying the river slope specification. Therefore, as long as the shape of centerline and valley slope remain the same, a certain “River Slope” specification always result in the amplitudes.